

Дж. Маджинис

**ПРОГРАММИРОВАНИЕ
НА СТАНДАРТНОМ
КОБОЛЕ**







JAMES B. MAGINNIS
Drexel University

**FUNDAMENTAL
ANSI
COBOL
PROGRAMMING**

Prentice-Hall, Inc.
Englewood Cliffs, N. J.
1975

Дж. Маджинис

ПРОГРАММИРОВАНИЕ НА СТАНДАРТНОМ КОБОЛЕ

Перевод с английского
С. Д. ЗЕЛЕНЕЦКОГО
и Л. А. ПОЗДНЯКОВА

под редакцией
В. И. СОБЕЛЬМАНА

ИЗДАТЕЛЬСТВО «МИР»
МОСКВА 1979

Вводный курс по изучению и применению широко распространенного языка программирования. Сначала описываются основные черты языка, образующие его ядро, затем более сложные средства языка: последовательная и произвольная выборка записей из файлов, сортировка, структурирование данных, обработка таблиц, сегментирование программ, передача сообщений, отладка программ. Дается большое количество примеров и упражнений, позволяющих глубоко и неформально усвоить язык и научиться пользоваться им.

Книга предназначена для программистов в первую очередь экономических задач и задач с большим объемом данных. Она может быть использована как учебное пособие для впервые изучающих этот язык и как справочник для владеющих им.

Редакция литературы по математическим наукам

2405000000

М $\frac{20004-029}{041(01)-79}$ 29-79

Original English language edition published by Prentice-Hall, Inc., Englewood Cliffs, New Jersey, U.S.A.
Copyright © 1975 by PRENTICE-HALL, INC.

© Перевод на русский язык, «Мир», 1979

Предисловие редактора перевода

За два десятилетия своего существования КОБОЛ получил широкое распространение во всем мире в качестве языка программирования задач обработки данных.

Основные отличительные особенности КОБОЛа по сравнению с другими языками программирования состоят в том, что, во-первых, он предусматривает обработку структур данных, во-вторых, в известном смысле он является подмножеством естественного языка и, в-третьих, в нем предпринята попытка решения проблемы независимости от специфики конкретных ЭВМ (проблемы совместимости программ для конструктивно различных ЭВМ).

Именно возможность оперировать с данными, организованными в древовидные структуры, делает КОБОЛ эффективным для обработки реальной информации из разных областей человеческой деятельности — информации, которая в большинстве случаев имеет сложную структуру. Другие языки программирования, ориентированные на обработку данных, как правило, копируют средства описания данных КОБОЛа. Примером может служить язык ПЛ/1, в который фактически перенесены без изменений средства описания данных КОБОЛа.

Поскольку КОБОЛ является подмножеством естественного языка, написанный на нем текст можно понимать без предварительного изучения правил КОБОЛа (правда, при условии, что составитель текста стремился быть понятным).

И, наконец, в КОБОЛе предусмотрен специальный раздел программы, в котором указывается вычислительное оборудование, используемое в программе. Транслятор, формируя программу в машинном коде, учитывает эти сведения.

Настоящая книга является подробным учебником по КОБОЛу. Отличительная ее особенность состоит в иллюстрации изложения большим количеством примеров, являющихся упрощенными моделями типовых задач сбора, коррекции и накопления данных в ЭВМ, учета материальных ценностей, начисления зарплаты, кадрового учета и пр. Книга содержит много упражнений, способствующих активному усвоению материала.

При переводе была сохранена английская нотация оригинала по двум причинам:

1) на большинстве ЭВМ в нашей стране сегодня используется версия КОБОЛа с английской нотацией (только недавно появилась возможность оснащения машин семейства ЕС ЭВМ транслятором с КОБОЛа с русской нотацией);

2) переход на русскую нотацию потребовал бы существенной переработки большинства примеров и упражнений, особенно связанных с упорядочением данных в алфавитном порядке, что вызвало бы значительный отход от оригинала.

За последние годы в нашей стране резко возрос интерес к КОБОЛу в связи с развитием работ по применению ЭВМ в различных областях экономики и управления народным хозяйством, в частности в связи с созданием автоматизированных систем управления (АСУ). Экономическая информация, как правило, имеет сложную структуру. Поэтому для задач ее обработки велика потребность в языке программирования, обладающем средствами оперирования со структурами данных.

Используемая в книге терминология полностью согласована с проектом Государственного стандарта «Язык программирования КОБОЛ». Этот стандарт предусматривает кроме русской и английскую нотацию, принятую в этом переводе. В тексте каждое зарезервированное в языке английское слово снабжено его русским эквивалентом, заключенным в скобки, при первом его упоминании и во многих других местах. В конце первой главы книги приводится сводная таблица английских зарезервированных слов и их русских эквивалентов.

В. И. Собельман

Предисловие

В 1959 г. для проведения стандартизации языков программирования экономических задач была создана специальная группа, получившая название рабочей группы КОДАСИЛ (CODASYL — COncference on DAta SYstems Language). В результате весьма успешной деятельности этой группы был разработан универсальный язык программирования, ориентированный на решение экономических задач, КОБОЛ (COBOL — COmmon Business Orien-ted Language), который в настоящее время является наиболее широко распространенным в мире языком данного типа. Этот язык признан и получил широкое распространение не только в Западном полушарии, но и в Японии и Западной Европе. В качестве эталона КОБОЛа группа разработала и опубликовала в своем журнале (CODASYL Journal of Development) описание синтаксиса и семантики языка.

В Соединенных Штатах Америки Национальное бюро стандартов (NBS — National Bureau of Standards) и Американский институт национальных стандартов (ANSI — American National Standards Institute) издали для правительственных учреждений и для предпринимателей, выпускающих вычислительное оборудование, стандарты КОБОЛа, основанные на рекомендациях КОДАСИЛ. Хотя между стандартами NBS и ANSI первоначально существовало некоторое расхождение, в настоящее время они полностью совпадают. Сегодня можно утверждать, что федеральный стандарт обработки информации (FIPS — Federal Information Processing Standard), изданный NBS для программирования на КОБОЛе, идентичен стандарту ANSI КОБОЛ.

За последние пятнадцать лет КОБОЛ существенно развился. Это развитие часто бывало спорным, однако сейчас КОБОЛ достиг разумной степени устойчивости, и им можно пользоваться на большинстве вычислительных машин. Компиляция, или перевод с КОБОЛа на машинные языки, осуществляется в настоящее время быстро и эффективно. КОБОЛ широко распространен; правительство США рекомендует использовать его во всех задачах, связанных с управлением, и большинство фирм следует этой рекомендации. В ближайшем будущем КОБОЛу предстоит еще большее распространение по мере расширения областей его применения.

Эта книга задумана в качестве учебника по синтаксису КОБОЛа и по программированию на нем. Основные понятия языка базиру-

ются на современном стандарте ANSI. Изложение сопровождается примерами и упражнениями, связанными с конкретными задачами обработки данных. Первые пять глав, в которых описываются основные возможности КОБОЛа, построены таким образом, чтобы читатель как можно скорее смог приступить к программированию реальных задач. Программирование — это искусство, которое базируется на практике, поэтому полезно, чтобы изучающий язык пытался выполнить написанные им программы на вычислительной машине.

В отдельных организациях эксплуатируются небольшие вычислительные машины, на которых нельзя реализовать все возможности языка, или используются старые компиляторы, не измененные в соответствии с современным стандартом языка. И сам язык, и его изложение в данной книге учитывают возможность таких ситуаций. Каждая программа на КОБОЛе содержит часть, называемую ENVIRONMENT DIVISION (РАЗДЕЛ ОБОРУДОВАНИЯ), которая должна быть написана пользователем и в точности описывает используемую машину. Кроме того, в первых пяти главах книги описание языка ограничивается теми возможностями, которые могут использоваться почти в любом компиляторе. Те же возможности, которые являются необязательными или представляют собой специальные расширения, относящиеся к конкретным реализациям языка, опущены. В главе 2 подчеркивается, что на некоторых вычислительных машинах могут использоваться специальные возможности. Хотя отклонения от стандарта, встречающимся в реальной практике, в данной книге уделяется лишь незначительное внимание, преподаватель должен помочь студентам заполнить подобные пробелы и сообщить им о специфических особенностях конкретной установки, которые, возможно, неизвестны оператору машины.

Глава I. Исходная программа

1.1. Введение в программирование на языке КОБОЛ

КОБОЛ является языком программирования, разработанным для решения экономических задач с использованием вычислительной техники. В учреждениях накапливаются данные о различных сторонах их деятельности, например таких, как производство или сбыт. Подобная информация обрабатывается с помощью вычислительной машины для составления отчетов, используемых в сфере управления. Кроме того, данные могут накапливаться в памяти вычислительной машины для их обработки в будущем, например для составления месячных сводок ежедневной продажи или для сравнения экономических показателей текущего года с показателями прошлых лет. Каждая единица информации, такая, как имя покупателя или количество пар обуви, проданное за день, называется *данным*. Над данными в ходе обработки могут выполняться действия трех типов: над числовыми данными могут производиться арифметические операции (сложение, вычитание, умножение и деление); числовые или буквенные данные могут сравниваться между собой, и результат такого сравнения может определять последующую обработку; данные могут перемещаться в памяти машины, подвергаясь при этом определенному редактированию.

Первый шаг в программировании на языке КОБОЛ состоит в анализе задачи, которую предстоит решать на вычислительной машине. В результате анализа производится разбиение всей задачи на более простые части (процедуры), с которыми может оперировать вычислительная машина. Такой анализ может быть облегчен благодаря использованию блок-схемы, представляющей собой диаграмму, показывающую последовательность, в которой следует выполнять процедуры.

Второй шаг в программировании на КОБОЛе состоит в *кодировании*, т. е. в записи процедурных шагов на языке КОБОЛ с тем, чтобы они стали доступны и понятны вычислительной машине. Настоящая книга описывает структуру и назначение языка КОБОЛ, а также технику кодирования различных процедур.

Третий шаг состоит в переводе и выполнении закодированной программы. Для этого потребуется использовать универсальную электронную вычислительную машину (ЭВМ), а также транслирующую программу, называемую *компилятором*. Конечным результатом трансляции должна стать последовательность машинных команд, которая при ее исполнении на ЭВМ выберет требуемые данные, обработает их с помощью заданного набора процедур и выработает новые данные для запоминания их в памяти ЭВМ и составления из них печатных отчетов.

Составление блок-схемы

Результат процедурного анализа задачи может быть изображен графически в виде блок-схемы. В блок-схеме используется лишь небольшое число основных элементов, которые могут сочетаться самыми различными способами.

При составлении блок-схемы все процедуры сводятся к операциям трех типов. Первая из них — операция обмена, по которой информация вводится в рабочую память машины, становясь доступной

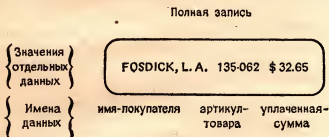


Рис. 1.1.

для обработки, или выводится из рабочей памяти. В блок-схеме эти операции идентифицируются путем помещения в ромбоид. Предусмотрены две операции обмена: одна с названием READ (ЧИТАТЬ) для ввода и другая с названием WRITE (ПИСАТЬ) для вывода. При вводе и выводе запись, являющаяся группой данных, передается как единое целое. На рис. 1.1 изображена запись, содержащая три данных: имя покупателя, артикул товара и уплаченную сумму. Во время чтения или записи все три данных передавались бы как единая группа, но вычислительная машина может оперировать с каждым данным отдельно. Суть операций считывания и записи описана в гл. 3 и более подробно в гл. 6, 7 и 10. Физически запись может быть печатной строкой, перфокартой или строкой символов на магнитной ленте.

Второй основной операцией является или арифметическая операция, или операция *перемещения* (т. е. внутренней пересылки данных). В блок-схеме шаги, соответствующие этим операциям, идентифицируются путем включения в прямоугольник. Операции перемещения описаны в гл. 3, арифметические операции — в гл. 4. Перемещения могут быть использованы для помещения данных в различные позиции и вставки специальных литер, таких, как валютные знаки и десятичные точки. С помощью арифметических операций могут производиться различные вычисления итогов или, например, расчет заработной платы.

Третья операция служит для выбора пути выполнения вычислений в зависимости от истинности или ложности заданного условия.

Различные условия описаны в гл. 4, а также в гл. 9. Простейшим примером такого условия может служить определение того, является ли числовое значение возраста какого-либо лица меньше 30 лет. Условие «возраст лица меньше 30» для всякого конкретного лица может быть либо точно истинным, либо точно ложным. В блок-схеме операция выбора пути идентифицируется заключением в ромб.

Помимо трех уже рассмотренных элементов блок-схемы: ромба для чтения и записи, прямоугольника для арифметических действий и перемещений и ромба для условного выбора пути, используются еще три элемента: стрелка, овал и окружность. Стрелка используется для соединения операций в блок-схеме. Она указывает следующее действие, например:



Конец стрелки указывает в схеме на операцию (блок), которая должна выполняться следующей (отсюда и название «блок-схема»). Процесс решения задачи, представленный блок-схемой, начинается в исходной точке и идет по пути, указанному стрелками, до некоторого конечного пункта. В ходе следования по этому пути безусловные шаги выполняются, а в случае условного выбора пути производится вычисление условий (ложь или истина), и в зависимости от результата вычисления выбирается одно из возможных направлений продолжения следования.

Начальные и конечные пункты заключаются в овалы. Название самого процесса в целом помещается в начальный овал, а в конечный овал или овалы (возможно несколько мест остановки) ставится слово «СТОП», например:



И наконец, небольшая окружность служит связкой, позволяющей опустить стрелки, которые могут сделать блок-схему запутанной и затруднить ее интерпретацию. Две окружности с одним и тем же символом внутри считаются связанными стрелкой независимо от того, как далеко они отстоят друг от друга в блок-схеме. Так фрагмент блок-схемы



эквивалентен фрагменту



Пример простой блок-схемы приведен на рис. 1.2. Программа предназначена для изъятия из *файла* устаревших записей. Файл представляет собой совокупность записей, созданных в разное время. Каждая запись содержит данные, значение которого представляет собой дату создания записи. Каждая запись содержит и другие элементы, которые не используются в данном случае. Согласно блок-схеме в самом начале выполнения программы в память машины считывается контрольная дата. (Из файла должны быть изъятые все записи с датой создания, более ранней, чем эта контрольная дата.) Затем записи данных последовательно считываются из старого основного файла. После того как каждая запись становится доступна вычислительной машине, прежде всего производится проверка, исчерпан ли весь файл (все ли записи считаны), и если нет, то проводится сравнение даты создания записи с контрольной датой. Если запись следует сохранить, то она переписывается в новый основной файл. Перед занесением в новый файл вся запись копируется. В противном случае считывается следующая запись старого основного файла, а текущая запись игнорируется. Таким образом, такая запись не попадает в новый основной файл. Два условия управляют последовательностью шагов при создании нового основного файла, из которого исключены все устаревшие записи. После завершения изъятия старый основной файл может быть уничтожен, а новый основной файл будет использоваться для дальнейшей обработки.

Процедурный анализ задачи может быть произведен с различной степенью детализации. Приведенный пример совсем прост. Однако

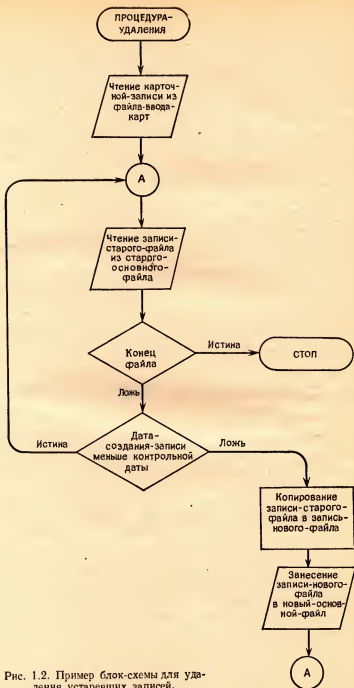


Рис. 1.2. Пример блок-схемы для удаления устаревших записей.

для написания программы и ее выполнения бывают необходимы дополнительные сведения. Одной из возможностей детализации могла бы служить проверка во время считывания на правильность и допустимость контрольной даты. Можно было бы разработать более подробную блок-схему, проверяющую, что значение месяца не превышает 12, значение дня — 31 (или точнее не превышает 28, 29 или 30 в зависимости от месяца и года). Можно было бы также проверять, не выходит ли значение года за разумные рамки (значение года 1066 должно вызывать подозрение для большинства файлов).

I. Файл-ввода-карт.

А. Состоит из одной записи, называемой карточной-записью и содержащей восемьдесят литер.

1. Запись данного типа содержит одно данное, называемое контрольной-датой, и ряд других данных, несущественных для рассматриваемого примера.

а. Данное контрольная-дата состоит из шести литер. Первые две литеры представляют собой год текущего столетия, следующие две цифры — месяц и, наконец, последние две цифры — день месяца. Например: 760225.

II. Старый-основной-файл.

А. Состоит из многих экземпляров записей одного типа, называемого записью-старого-файла. Записи этого типа содержат 456 литер.

1. Запись-старого-файла состоит из нескольких данных, одно из которых называется датой-создания-записи.

а. Данное дата-создания-записи состоит из шести литер и имеет тот же самый формат, что и описанное выше данное контрольная-дата. Остальные данные не важны.

III. Новый-основной-файл.

А. Состоит из многих экземпляров записей одного типа, называемого записью-нового-файла. Формат и длина записи-нового-файла такие же, как и у записи-старого-файла. Выходные записи будут копиями входных записей, у которых дата-создания-записи не меньше, чем контрольная-дата.

Рис. 1.3. Образец описания файлов для примера удаления записей.

В случае появления какого-либо из этих условий может быть напечатано сообщение об ошибке и выполнение программы остановлено до тех пор, пока человек не решит, следует ли не принимать во внимание допущенную ошибку или нужно ввести новую дату.

В дополнение к шагам по выполнению программы, показанным в блок-схеме, в ходе анализа должны быть выяснены дополнительные сведения, необходимые для составления программы. Так, понадобятся сведения о том, где находится старый основной файл и как должен быть организован к нему доступ. Также должен быть описан формат записи, т. е. способ расположения в ней данных, и должно быть приведено точное описание каждого данного. Напри-

мер, каким образом в каждой записи представлена дата — как месяц, день и год или как год, месяц и день? Представлена ли дата в таком виде, как АПРЕЛЬ 15, 1976, или записана цифрами, как 760415? Программа для машины может быть составлена почти для любой ситуации, но ситуация должна быть определена точно и должны быть заданы форматы всех записей. Пример ограниченного описания программы исключения записей из файла дан на рис. 1.3. Каждый элемент данных, каждая запись и каждый файл, которые обрабатываются программой на КОБОЛе, должны иметь имя с тем, чтобы на них можно было ссылаться. Присвоение этих имен осуществляет программист, и они обычно выбираются с определенным mnemonicским, или смысловым, значением. В нашем примере файл ввода карт будет содержать всего одну запись, но тем не менее это файл и он должен иметь свое имя. Тип записи, содержащейся в файле-ввода-карт, называется карточной-записью. Эта запись состоит из восьмидесяти литер, но только одно данное, называемое контрольной-датой, представляет интерес в нашем случае. Контрольная-дата содержит значение даты, которое будет использоваться для выборочного исключения записей из старого-основного-файла. Старый-основной-файл содержит только один тип записей, называемый записью-старого-файла, но может состоять из многих экземпляров записей. (Экземпляр записи — это фактическая реализация подобной записи со своими уникальными данными. В расчетах заработной платы файл будет содержать один экземпляр записи, называемой платежной-записью, на каждого служащего.) В нашем примере каждая запись будет содержать такое данное, как дата-создания-записи, а также и другую информацию (как показано, запись имеет длину 456 знаков), но другие данные не представляют для нас интереса. Новый файл, называемый новым-основным-файлом, будет создан в результате работы программы. Новый-основной-файл содержит много экземпляров записей, называемых записями-нового-файла, которые будут являться копиями неисключенных записей-старого-файла.

Запись программы на языке КОБОЛ

Второй шаг в программировании на КОБОЛе состоит в кодировании, или записи процедурных шагов на этом языке. Действия современной вычислительной машины управляются программой, которая состоит из числовых команд, находящихся в памяти машины и выполняемых по одной блоком управления. Подготовка программы на языке машины является делом трудоемким, длительным и сопряженным с возможностями ошибок. КОБОЛ является языком, облегчающим подготовку программы на языке машины. Важно сознавать, что программа на КОБОЛе не является программой, которая непосредственно управляет работой вычислительной машины.

Прикладная программа на КОБОЛе, написанная пользователем, преобразуется специальной компилирующей программой в программу на языке машины, которая, будучи вызванной в память, управляет работой машины. Использование КОБОЛа исключает многие детали и предоставляет пользователю определенные средства, позволяющие локализовать смысловые ошибки с помощью диагностических сообщений. Программа на языке КОБОЛ должна быть составлена точно и ясно. Соблюдение грамматических правил языка КОБОЛ является необходимым для избежания ошибок при программировании.

Грамматические правила КОБОЛа были установлены не произвольно, а так, чтобы позволить пользователю достаточно просто и вместе с тем точно описывать необходимые действия. Однако в некоторых случаях у пользователя может появиться желание использовать другие формы выражения или устранить некоторые ограничения. Начинаящим, в особенности, мы советуем отказаться от этого желания и не терять времени, пытаясь обойти синтаксические ограничения. Язык был создан как язык управления машиной и требует высокой степени точности, не свойственной естественным языкам. Он предназначался для составления исходных программ, которые могут быть переведены на языки многих различных машин. С этой целью в нем игнорируются многие индивидуальные особенности вычислительных машин.

Еще одним полезным свойством КОБОЛа является то, что он обеспечивает легкость понимания программы, что облегчает возможность последующих модификаций. Эту черту самодокументируемости многие считают одним из важнейших свойств КОБОЛа. В области экономики ситуация, которая привела к возникновению конкретной задачи, не остается неизменной, так как меняются условия производства и сбыта, выходят новые постановления руководящих органов, которые должны учитываться. Таким образом, программа, написанная в январе в соответствии с существовавшими в то время требованиями, может нуждаться в изменении уже в июне. В силу необходимости динамических изменений программы требуется очень тщательно описывать первоначальную программу, чтобы иметь возможность затем вносить в нее изменения. Тщательность документирования частично достигается с помощью блок-схемы и описания файлов, но, кроме того, и с помощью самого языка КОБОЛ, использующего фразы, ясно описывающие для читателя требуемые действия.

Пример точной программы на КОБОЛе показан на рис. 1.4. Вряд ли на этой стадии изучения языка все части такой программы будут предельно ясны, но приведенный пример демонстрирует результат второй фазы программирования. Целью программы является выполнение операции чистки файла, упомянутой выше. Идентификатором программы является имя EXAMPLE (ПРИМЕР). За-

IDENTIFICATION DIVISION.
PROGRAM-ID. EXAMPLE.

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.
SOURCE-COMPUTER. MARK-I,
OBJECT-COMPUTER. MARK-I.

INPUT-OUTPUT SECTION.

FILE-CONTROL.
SELECT CARD-INPUT-FILE ASSIGN TO CARD-READER.
SELECT OLD-MASTER-FILE ASSIGN TO MAGNETIC-TAPE.
SELECT NEW-MASTER-FILE ASSIGN TO MAGNETIC-TAPE.

DATA DIVISION.

FILE SECTION.

FD CARD-INPUT-FILE
LABEL RECORD IS OMITTED.
01 CARD-RECORD.
05 DISCARD-DATE PICTURE IS XXXXXX.
05 FILLER PICTURE IS X(74).

FD OLD-MASTER-FILE
LABEL RECORD IS STANDARD.
01 OLD-FILE-RECORD.
05 CREATION-DATE PICTURE IS XXXXXX.
05 FILLER PICTURE IS X(450).

FD NEW-MASTER-FILE
LABEL RECORD IS STANDARD.
01 NEW-FILE-RECORD PICTURE IS X(456).

PROCEDURE DIVISION.

I. OPEN INPUT CARD-INPUT-FILE OLD-MASTER-FILE.
OPEN OUTPUT NEW-MASTER-FILE.

READ CARD-INPUT-FILE RECORD
AT END STOP RUN.

A. READ OLD-MASTER-FILE RECORD
AT END STOP RUN.
IF CREATION-DATE IS LESS THAN DISCARD-DATE
GO TO A.
WRITE NEW-FILE-RECORD FROM OLD-FILE-RECORD.
GO TO A.

Рис. 1.4. Пример КОБОЛ-программы для удаления старых записей.

метьте, что все объекты в языке КОБОЛ должны иметь имена, например: имя программы, имя файла, имя записи, имя данных. Трансляция программы должна быть осуществлена на вычислительной машине MARK-1, которая является исходной машиной, а выполняться программа должна на машине MARK-1, являющейся также и рабочей машиной. MARK-1 — это название (имя) машины, созданной отделом обработки данных UNIVAC корпорации Sperry Rand. Термин «исходная машина» относится к машине, переводящей с языка КОБОЛ на машинный язык. Обычно это будет та же самая машина, которая используется для выполнения программы, т. е. рабочая машина. В некоторых случаях это будут две различные машины: одна для перевода, а другая для исполнения. В рассматриваемой задаче используются три файла. Они называются CARD-INPUT-FILE (ФАЙЛ-ВВОДА-КАРТ), OLD-MASTER-FILE (СТАРЫЙ-ОСНОВНОЙ-ФАЙЛ) и NEW-MASTER-FILE (НОВЫЙ-ОСНОВНОЙ-ФАЙЛ). Имена для файлов выбираются программистом и имеют силу только в пределах этой программы. Первый файл связан с устройством считывания перфокарт, а два других — с лентопротяжными устройствами. Описание устройств машины, таких, как устройство считывания перфокарт и лентопротяжное устройство, будет дано в следующей главе. В секции FILE SECTION (СЕКЦИЯ ФАЙЛОВ) раздела DATA DIVISION (РАЗДЕЛ ДАННЫХ) описаны форматы записей в файлах. Каждая часть FD (Описание файла) описывает один файл. Определено, что запись CARD-RECORD имеет 80 позиций литер. Первые шесть позиций (PICTURE IS XXXXXX) названы DISCARD-DATE (КОНТРОЛЬНАЯ-ДАТА). Остальная часть записи, занимающая 74 позиции, названа FILLER (ЗАПОЛНИТЕЛЬ). Это означает, что размещенные в этих позициях данные не используются (безразличны) в данной программе.

Файл OLD-MASTER-FILE состоит из записей, содержащих в первых шести позициях данное CREATION-DATE (ДАТА-СОЗДАНИЯ). Эти записи имеют 456 позиций, и последние 450 позиций (PICTURE IS X(450)) могут содержать произвольные данные, например имя, адрес, расчетную информацию и т. п., но не представляют интереса для данной программы и не обозначаются именами. Если какой-либо элемент не имеет имени, то в программе на него нельзя сослаться. Слово FILLER не имя, а лишь указание на занятое место. В записи NEW-FILE-RECORD нет иных имен, кроме имени всей записи, так как отдельные ее части не будут обрабатываться. Каждая такая запись представляет собой лишь копию некоторой записи OLD-FILE-RECORD.

Раздел PROCEDURE DIVISION (РАЗДЕЛ ПРОЦЕДУР) содержит приказы, которые машина должна исполнять. В этом разделе есть две процедуры, названные именами I и A. Процедура I открывает для обработки три файла, два из которых предназначены для

операций ввода (INPUT) и один — для вывода (OUTPUT), и считывает первую и единственную запись из файла CARD-INPUT-FILE, которая содержит значение данного DISCARD-DATE. Процедура А читает записи из файла OLD-MASTER-FILE и переписывает в файл NEW-MASTER-FILE те из них, которые были созданы после контрольной даты, т. е. те, для которых не выполнено условие CREATION-DATE IS LESS THAN DISCARD-DATE (ДАТА-СОЗДАНИЯ МЕНЬШЕ КОНТРОЛЬНАЯ-ДАТА). Если какой-либо файл ввода достигает конца, программа выполнит оператор STOP RUN (ОСТАНОВИТЬ РАБОТУ) и завершится. В реальной программе неожиданный конец файла CARD-INPUT-FILE обычно приводит к печати сообщения с тем, чтобы пользователь знал, что работа прервалась преждевременно. Оператор GO TO A (ПЕРЕЙТИ К А) осуществляет передачу управления, как показано на блок-схеме рис. 1.2. В противном случае операции выполняются в том порядке, в котором они записаны.

Этот пример демонстрирует некоторые из преимуществ КОБОЛа. Первое — это совместимость, которая представляет собой возможность использовать программу на различных вычислительных машинах с минимальными изменениями. С помощью изменения только раздела ENVIRONMENT DIVISION (РАЗДЕЛ ОБОРУДОВАНИЯ) эта программа может быть скомпилирована и выполнена на другой вычислительной машине. Второе преимущество — документированность, согласно которой программа просто и ясно указывает, что необходимо последовательно выполнить операторы READ (ЧИТАТЬ), IF (ЕСЛИ) и WRITE (ПИСАТЬ). Типы форматов и записей точно определены в секции FILE SECTION, и когда пользователь в достаточной степени овладеет КОБОЛом, он сможет по ним узнать значительно больше о самих файлах. Третьим преимуществом КОБОЛа является более простое и быстрое программирование, поскольку в этом языке отсутствуют подробности, связанные с особенностями используемой машины. В силу этого сокращается время от постановки задачи до выдачи решения машиной как при составлении программы, так и при внесении дальнейших изменений. Четвертое преимущество процесса программирования на КОБОЛе, которое не отражено в данном примере, заключается в возможности выдачи во время компиляции диагностических сообщений, которые помогают пользователю обнаружить недопустимые операции, орфографические и грамматические ошибки.

Процесс трансляции

Третьим этапом программирования на КОБОЛе является трансляция и выполнение программы. Этот этап включает много действий, большинство из которых не может быть описано подробно, поскольку они уникальны для каждой отдельной установки, ис-

пользуемой для компиляции и выполнения программы. После того как КОБОЛ-программа написана, она должна быть представлена в какой-либо доступной для вычислительной машины форме. Обычно это делается путем перфорации на картах, но возможна запись на магнитную ленту или диск при помощи телетайпов или специальных клавишных устройств. Затем программа проходит синтаксическую проверку, в ходе которой транслятор пытается создать версию КОБОЛ-программы на языке машины. Эти попытки часто терпят неудачу из-за того, что программист нарушил какое-либо синтаксическое (грамматическое) правило, например пропустил точку там, где она нужна, или поставил пробел там, где это не разрешено. Нередко допускаются ошибки при перфорации, например перфорация цифры 1 вместо буквы I или цифры 2 вместо буквы Z и т. д.

После нескольких неудачных попыток, вызывающих недовольство пользователя, исходная программа, наконец, становится приемлемой для машины, и составляется рабочая программа. Рабочая программа — это машинная версия исходной КОБОЛ-программы, предназначенная для рабочей машины, упомянутой в разделе ENVIRONMENT DIVISION. Однако программа, как правило, не будет сразу же работать даже в простых случаях из-за логических ошибок в последовательности операторов или из-за наличия в программе операторов, которые приказывают машине выполнять неправильные действия. Такие ошибки выявляются в ходе *отладки* программы. Отладка программы состоит в пробном исполнении программы на рабочей машине над специально подготовленными тестовыми данными. Расчет и подготовка тестовых данных для программы могут потребовать много времени. Данные следует выбирать таким образом, чтобы установить, работает ли программа, и проверить все части всех процедур. Это можно проиллюстрировать простым примером: два и два не подходят для проверки умножения, так как оператор ADD (СЛОЖИТЬ), написанный по ошибке вместо умножения, также даст в ответе четыре. Когда данные считываются программой и выдается неправильный ответ, программист проводит поиск ошибок, их исправление и дальнейшую проверку программы, пока не будет создана *работающая* программа. Работающая программа — это такая программа, которая предположительно будет работать, как задумано, хотя практически никогда не может быть строго доказано, что она будет работать правильно. На этой стадии просто считается, что программа работает правильно («Она еще не обманула наших ожиданий»).

Компиляция, загрузка и выполнение КОБОЛ-программы осуществляются под управлением другой программы, называемой операционной системой (или монитором, или супервизором, или основной управляющей программой). Большим вычислительным комплексом управляет не человек, а специальная управляющая программа. Операционная система получает директивы в форме языка управле-

альную карту компиляции, загрузки и передачи управления. При этом карты описания стандартного файла могут отсутствовать.

На рис. 1.5 показана схема, описывающая действия, обеспечивающие успешное программирование задачи на машине. Эти действия начинаются с четкой формулировки задачи. На практике на этот этап тратится большая часть времени от начала до получения окончательной рабочей программы, так как фаза формулировки является решающей. Большинство реальных ошибок при решении задач на вычислительной машине обусловлены тем, что задачи сформулированы нечетко или неправильно. Анализ задачи требует от программиста сообразительности, особенно для выработки наилучшего решения. Следующий шаг — запись решения на языке КОБОЛ. Затем КОБОЛ-программа вводится в машину для компиляции, и этот процесс повторяется до тех пор, пока компиляция не будет удачной. Это не значит, что программа будет работать или выдавать правильные результаты: это просто значит, что программа представляет собой правильную запись на языке; ведь возможны еще и логические ошибки. Фаза выполнения требует исходных данных, которые либо специально создаются для пробного прогона, либо уже имеются в вычислительном центре. Полученный результат фазы выполнения может заставить программиста пересмотреть программу и начать процесс компиляции снова. В конце концов результат, полученный после выполнения программы, покажется правильным, и пользователь может прийти к выводу (возможно ошибочному), что программа работает. Выбор данных для «доказательства правильности» программы является нелегкой задачей. Конечный этап не показан на блок-схеме. Это этап, на котором программа готовится для эксплуатации. На этапе подготовки к эксплуатации возникают проблемы обучения операторов, планирования работы, подготовки лент и перфокарт и распространения программной документации.

Пример

Предположим, что существует файл записей, в котором каждая запись содержит поле с числовой величиной. Составьте блок-схему, описывающую процедуру вычисления среднего значения всех этих величин в файле и запишите результат в другой файл.

Решение

Описание файла:

Имя файла — существующий-файл. Этот файл содержит много экземпляров записей типа, который называется существующая-запись. При этом каждая запись содержит элемент данных, называемый поле-величины.

Блок-схема: см. рис. 1.6.

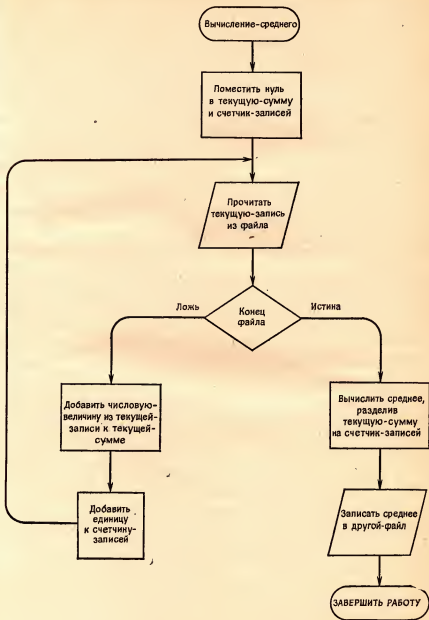


Рис. 1.6.

Пример

Допустим, что существуют два файла, называемые A-FILE и B-FILE. Первый файл состоит из записей, называемых A-RECORD и содержащих среди прочего данное, называемое A-ITEM. Файл B-FILE содержит запись точно такого же формата, за исключением того, что соответствующее данное называется B-ITEM. Каждый файл упорядочен по возрастанию величины данного A-ITEM и данного B-ITEM, т. е. величины данных A-ITEM возрастают в каждой последующей записи файла A-FILE и то же самое справедливо для данных B-ITEM в файле B-FILE. В такой ситуации говорят, что файлы упорядочены по этим данным. Составьте блок-схему для процедуры образования нового файла, называемого C-FILE, который представляет собой копию всех записей файла A-FILE, за исключением тех, для которых имеется запись в файле B-FILE, чье данное B-ITEM равно данному A-ITEM в файле A-FILE. В таком случае перенесите запись B-RECORD в файл C-FILE и пропустите запись A-RECORD. Внутри каждого файла не допускаются одинаковые величины. Они могут встретиться только в разных файлах. Возможно, что попадутся величины данных B-ITEM, которые вообще не равны ни одному данному A-ITEM. Пропустите подобные записи, когда они встретятся, и продолжайте поиски одинаковых величин. Ниже приводится пример того, как могут выглядеть подобные файлы:

A-FILE (содержит записи, именуемые A-RECORD)

A-ITEM	Остаток записи
35	Остаток записи один в A-FILE
40	Остаток записи два в A-FILE
42	Остаток записи три в A-FILE
50	Остаток записи четыре в A-FILE
55	Остаток записи пять в A-FILE
62	Остаток записи шесть в A-FILE

B-FILE (содержит записи, именуемые B-RECORD)

B-ITEM	Остаток записи
25	Остаток записи один в B-FILE
35	Остаток записи два в B-FILE
45	Остаток записи три в B-FILE
55	Остаток записи четыре в B-FILE
65	Остаток записи пять в B-FILE

Результат должен быть следующим:

C-FILE (содержит записи, именуемые C-RECORD)

C-ITEM

Остаток записи

35	Остаток записи два в B-FILE
40	Остаток записи два в A-FILE
42	Остаток записи три в A-FILE
50	Остаток записи четыре в A-FILE
55	Остаток записи четыре в B-FILE
62	Остаток записи шесть в A-FILE

Решение

Блок-схема: см. рис. 1.7.

Описание

Файл A-FILE содержит записи, называемые A-RECORD, которые включают данное A-ITEM.

Файл B-FILE содержит записи, называемые B-RECORD, которые включают данное B-ITEM.

Файл C-FILE будет содержать записи, называемые C-RECORD, являющиеся копиями либо записей A-RECORD, либо записей B-RECORD.

Упражнения

1. Каковы преимущества языка КОБОЛ?
2. Назовите процессы, которые выполняются при помощи КОБОЛа на ЭВМ?
3. Составьте блок-схему для определения процедуры хранения и последовательной перезаписи в другой файл записи данных из старого-основного-файла-записей, которая содержит самую последнюю дату-записи.
4. Сопоставьте пронумерованные понятия с определениями, помеченными буквами:

Понятие

Определение

- | | |
|----------------|---|
| (1) условие | (а) граф последовательности процедур |
| (2) блок-схема | (б) одна или больше литер, обрабатываемых как единица данных |
| (3) данное | (в) способ расположения группы данных |
| (4) файл | (г) набор записей |
| (5) формат | (д) ситуация, для которой во время выполнения может быть определено значение истинности |

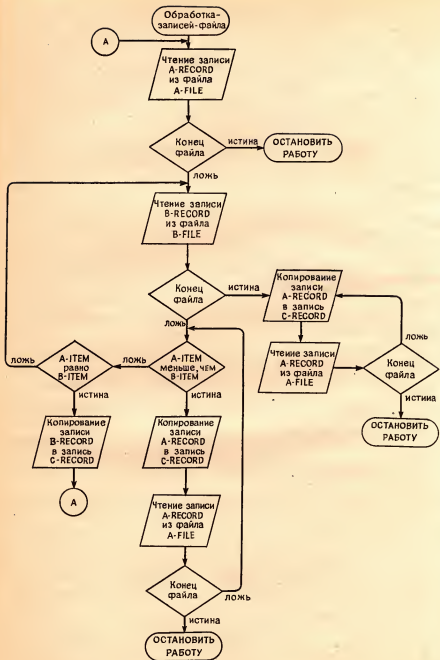


Рис. 1.7.

1.2. Языковой формат

Пользователь должен составлять программы на КОБОЛе в соответствии с определенными правилами. При написании программы может быть использован только определенный набор символов (литер). Например, в этот набор не входят строчные буквы. Из литер составляются *слова*, применяемые для наименования объектов (файлов, записей и т. д.) или вызывающие действия вычислительной машины (READ — ЧИТАТЬ, MOVE — ПОМЕСТИТЬ и т. д.). Кроме того, некоторые строки литер, называемые *литералами*, имеют в языке фиксированные значения, определяемые набором литер, из которых они составлены, например 1.25 (единица, за которой следует десятичная точка, далее двойка и литера пять) имеет значение один с четвертью, когда рассматриваемая строка используется в арифметических выражениях. Из слов в свою очередь составляются фразы и предложения, а они в конечном счете объединяются в программу, приведенную на рис. 1.4.

Для того чтобы компилятор перевел КОБОЛ-программу в программу на языке машины, исходная КОБОЛ-программа должна быть представлена машине каким-то приемлемым для нее способом. Этот приемлемый для нее способ будет различным для различных вычислительных машин. Для некоторых машин исходную программу необходимо будет подготовить на перфокартах. Пример набора перфокарт, подготовленных для обработки КОБОЛ-компилятором, приведен на рис. 1.8. Однако для некоторых машин КОБОЛ-программа вводится в машину через телетайп или клавишный пульт. Для сохранения общности КОБОЛа любая строка КОБОЛ-программы так и называется *строка* программы, а не карта, запись или как-то по-другому. Таким образом, КОБОЛ-программа представляет собой последовательность строк, причем каждая строка состоит из слов, составленных из литер.

Литеры КОБОЛа

Каждый язык имеет свой собственный основной набор символов. В элементарной арифметике используются цифры от 0 до 9 и некоторые знаки операций, такие, как плюс или минус. Алфавит греческого языка состоит из букв — альфа, бета и т. д., в то время как письменный английский имеет строчные и заглавные буквы, которые могут быть объединены в слова и предложения. Короче говоря, языки имеют основные символы, которые могут объединяться в соответствии с правилами языка для выражения «смысла». Язык КОБОЛ имеет свой собственный основной набор символов (называемых в КОБОЛе литералами), и только их можно использовать для написания программы на этом языке. Одной из главных причин ограниченности допустимого набора литер являются ограничения, налагае-

мые устройствами ввода машин. Одно время вычислительные машины могли воспринимать только числовые значения, позже были сконструированы машины, распознающие и читающие также и буквенные литеры. Хотя многие машины в настоящее время различают до 256 различных литер, такой способностью обладают еще не все машины. Число литер КОБОЛа ограничено семьюдесятью одной литерой¹⁾. *Литера* определяется как основной и неделимый элемент языка. Набор литер КОБОЛа приводится ниже:

<i>Литеры</i>	<i>Значения</i>
0, 1 ... 9	цифра
A, B, ..., Z, ..., Я	буква (только прописные буквы)
	пробел
+	плюс
—	минус или дефис
*	звездочка
/	дробная черта (слэш)
=	знак равенства (равно)
\$	валютный знак
,	запятая
;	точка с запятой
.	точка (десятичная точка)
"	кавычки
(круглая скобка левая
)	круглая скобка правая
>	больше
<	меньше

Набор литер КОБОЛа может быть разделен на группы. *Цифровая литера* — это одна из десяти цифр. *Буквенная литера* — это одна из сорока шести заглавных букв или пробел (что составляет сорок семь литер). *Литера пунктуации* — это одна из следующих:

,	запятая
;	точка с запятой
.	точка
"	кавычки
(круглая скобка левая
)	круглая скобка правая
/	дробная черта

¹⁾ В русской нотации КОБОЛа в набор литер языка включены прописные буквы русского алфавита, не совпадающие по начертанию с латинскими буквами. — *Прим. перев.*

Литера отношения — это один из символов больше, меньше или равно. Наконец, *специальная литера* — это любая литера, кроме числовых и буквенных. Таким образом, специальные литеры включают литеры отношения и пунктуации, а также \$, *, —, и +, но они не включают пробел. Только эти литеры, и никакие другие, могут использоваться для составления строки КОБОЛ-программы. Однако для изображения данных, которые машина может считать, обработать и выдать, набор литер не ограничивается этой 71 литерой. В зависимости от машины КОБОЛ-программа может обрабатывать вплоть до 256 различных символов, хотя сама программа должна быть *записана* с помощью всего 71 литеры. Это значит, что на некоторых машинах можно обрабатывать заглавные и строчные буквы и специальные символы, такие, как @, #, %, \$ и т. д. Все символы, которые попадают в набор для обработки любой данной машины, называются набором буквенно-цифровых литер.

Литеры КОБОЛа могут быть расположены в смежной последовательности (не содержащей пробелов) для формирования слов и литералов. Примером смежной последовательности будет КОТ; не-смежная последовательность содержит промежуточные пробелы, например К О Т. Объекты в смежной последовательности называют смежными, как в предложении: «сорок восемь смежных штатов США, исключая Аляску и Гавайи». Слово КОБОЛа — это смежная последовательность, содержащая не более чем тридцать литер, включая цифры, буквы или дефис (исключая случаи, когда дефис оказывается в начале или в конце слова). Слова отделяются пробелами или знаками препинания. Примеры слов:

CARD-INPUT-FILE
AIB2C3
SOURCE-COMPUTER

Следующие слова неправильны:

-START-
\$\$MAN
JOHN/JIM
MAN.HOME

Первое слово неверно, так как дефис не может стоять в начале и в конце; остальные три неверны, так как они включают другие специальные символы, кроме дефиса. Поскольку слова разделяются пробелом, допустимое слово не может включать пробел.

Некоторые слова имеют специфические значения в языке КОБОЛ и не могут использоваться программистом в качестве имен. Они называются *зарезервированными словами* — это такие слова, как ADD (СЛОЖИТЬ), DIVIDE (РАЗДЕЛИТЬ), ASCENDING (ВОЗРАСТАНИЮ), REWIND (ПЕРЕМОТКИ), PROGRAM-ID

(ПРОГРАММА). Имеется большое количество зарезервированных слов, и каждое из них имеет определенное значение в языке КОБОЛ. Использование любого из них в качестве имени (например, имени файла, записи, данных) приводит к синтаксической или логической ошибке. Список зарезервированных слов приведен в конце данной главы. Зарезервированных слов так много, что трудно случайно не использовать какое-либо из них. Однако не многие из зарезервированных слов имеют более одного дефиса, и ни одно из них не оканчивается на -X, -Y, или -Z. Таким образом, слова типа CARD-INPUT-FILE с двумя дефисами или JOHN-X с одной конечной литерой после дефиса являются вполне приемлемыми именами. Не существует зарезервированных слов, состоящих из одной буквы, типа A или B. Для экономии места в этой книге в некоторых примерах будут использоваться имена из одной буквы. Программисты, однако, должны избегать однобуквенных слов, но не потому, что они не верны, а потому, что выбранные имена должны быть как можно более описательными. Использование описательных имен значительно поможет в реальной ситуации во время модификации программ.

Наиболее важными знаками пунктуации в языке КОБОЛ в начале изучения языка являются точка и кавычка. Точка может быть использована двояко: как десятичная точка и как указатель конца. Точка-указатель конца заканчивает предложение, статью или заголовок. Хотя точке может предшествовать пробел, обычно такая возможность не используется. За точкой (не десятичной точкой) всегда должен следовать по крайней мере один пробел. Другими знаками пунктуации являются круглые скобки, запятая, точка с запятой. Их использование будет описано в последующих главах.

Литералы КОБОЛА

Литерал в КОБОЛе — это смежная строка литер, значение которой определяется литерами этой строки. Например, 1.25 имеет значение один с четвертью. Литералы бывают двух типов — числовые и нечисловые. Числовые литералы состоят из одной или нескольких цифр и могут включать в себя десятичную точку или алгебраический знак, который, если он есть, должен быть самой левой литерой. Крайняя правая десятичная точка будет всегда истолкована как знак пунктуации — точка. Если необходимо представить числовое значение 35., оно должно быть записано как 35.0. Нечисловой литерал — это строка литер, ограниченная кавычками. Строка внутри кавычек может содержать любой символ из числа воспринимаемых машиной (не только 71 литеру КОБОЛа), но не может содержать кавычки.

Литералы используются или в арифметических операциях, или для присваивания значений данным; например, оператор:

ADD 1.5 TO ITEM-X

добавит к данному ITEM-X 1.5; оператор:
MOVE 0.0 TO FLAG-B

присвоит переменной FLAG-B значение 0.0; оператор:
MOVE "NAME AND ADDRESS" TO ITEM-G

присвоит в качестве значения данному ITEM-G строку "NAME AND ADDRESS". Значения данных ITEM-X и FLAG-B числовые, а значение данного ITEM-G нечисловое. Значение данного ITEM-G само не содержит кавычки, которые являются ограничителями. Другой пример использования нечислового литерала:

DISPLAY "HELLO OUT THERE" ON PRINTER.

Еще примеры литералов:

—1.250
16000
123.0008
.075
"REPORT ON SALES"

Следующие литералы неверны:

1.24CR
\$12.60
125.
300.0—
"SAY"GOOD-BYE"TO ME"

так как в числовом литерале не могут появляться литеры, кроме цифр, десятичной точки или знака, и, кроме того, знак должен быть крайним слева, а десятичная точка не может быть крайней справа. Последний нечисловой литерал также неверен, так как кавычки не могут быть заключены в кавычки.

Фразы и операторы

Слова КОБОЛА, как зарезервированные, так и определенные пользователем, и литералы объединяются во *фразы* и *операторы*. Фразы носят описательный характер, а операторы — повелительный. Фраза является последовательностью слов, предназначенной для определения свойства некоторого объекта. Например, фраза

LABEL RECORDS ARE STANDARD

определяет, что метки (LABEL RECORDS) файла (см. рис. 1.4) являются стандартными. Операторы состояются из слов и сим-

волов. Они всегда начинаются с глагола. Глагол является одним из зарезервированных слов, выражающих действие, например: ADD (СЛОЖИТЬ), DIVIDE (РАЗДЕЛИТЬ), GO (ПЕРЕЙТИ), MOVE (ПОМЕСТИТЬ), OPEN (ОТКРЫТЬ), CLOSE (ЗАКРЫТЬ), WRITE (ПИСАТЬ).

Примеры операторов:

```
GO TO SEARCH-PROCEDURE
SUBTRACT 1.9 FROM RESULT-G
MOVE ABLE-X TO ANSWER-X
```

Фразы и операторы никогда не появляются вместе, так как фразы описательны по своей природе, а операторы задают выполнение действий. Однако фразы могут объединяться с фразами, а операторы с операторами. Любое количество фраз описания, заканчивающихся точкой, называется *статьей*. Любое количество операторов, заканчивающихся точкой, называется *предложением*.

Пример статьи:

```
FD OLD-MASTER-FILE LABEL RECORD IS STANDARD.
```

Пример предложения:

```
READ OLD-MASTER-FILE AT END MOVE "FINISH" TO
ITEM-X GO TO P-1.
```

Во многих случаях предложение состоит всего из одного оператора, заканчивающегося точкой, такого, как

```
GO TO P-10.
```

Статьи и предложения должны быть сгруппированы в *параграфы*. Параграф состоит из имени-параграфа, за которым следует точка, и следующих за ней одной или более статей или предложений. Имя-параграфа и завершающая его точка называются заголовком-параграфа. Таким образом, параграф состоит из заголовка-параграфа и его тела, как, например:

```
OPEN-FILES-PARAGRAPH.
OPEN INPUT CARD-INPUT-FILE OLD-MASTER-FILE.
OPEN OUTPUT NEW-MASTER-FILE.
```

В этом примере тело состоит из двух предложений, каждое из которых заканчивается точкой.

Однако организационная структура КОБОЛ-программы на этом не заканчивается. Параграфы могут группироваться, образуя *секции*. В свою очередь секции объединяются, образуя *разделы*, и, наконец, точно четыре раздела составляют КОБОЛ-программу. Таким образом, получается всего одна программа, и она венчает пирамиду группировок, начинающихся с базовых литер. На рис. 1.9



Рис. 1.9. Иерархическая структура КОБОЛа.

схематично показана эта структура. Ниже приводится один из четырех разделов программы:

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER.

MARK-1.

OBJECT-COMPUTER.

MARK-1.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

SELECT CARD-INPUT-FILE ASSIGN TO CARD-READER;

SELECT OLD-MASTER-FILE ASSIGN TO MAGNETIC-TAPE.

Раздел оборудования (ENVIRONMENT DIVISION) делится на две секции: секцию конфигурации (CONFIGURATION SECTION) и секцию ввода-вывода (INPUT-OUTPUT SECTION). Заголовок первого параграфа секции конфигурации — SOURCE-COMPUTER (ИСХОДНАЯ-МАШИНА). Единственная статья имеет только одну фразу MARK-1. Параграф FILE-CONTROL (УПРАВЛЕНИЕ-ФАЙЛАМИ) в секции ввода-вывода имеет две статьи, каждая из которых состоит из нескольких фраз.

Строки КОБОЛА

Все предложения, статьи, разделы и т. д. должны быть записаны в виде строк на некотором носителе. Для программиста этим носителем является бумага, но для вычислительной машины это должен быть носитель, воспринимаемый ею, такой, как карты или телетайпная лента. В каждой строке имеется конечное число позиций для записи литер; каждая машина может иметь свой собственный носитель ввода, но для каждой из них должно сохраняться понятие строки и позиции литеры в строке. Понятие строки и позиции литер используется при определении стандартного формата для ввода КОБОЛ-программы в машину. Строка подразделяется на *поля*, каждое из которых ограничено *отметками*. В строке устанавливается пять отметок:

Отметка L — предшествует самой левой позиции литеры в строке.

Отметка C — находится между шестой и седьмой позицией литеры в строке.

Отметка A — находится между седьмой и восьмой позицией литеры в строке.

Отметка В — находится между одиннадцатой и двенадцатой позицией литеры в строке.

Отметка R — непосредственно следует за самой правой позицией литеры в строке.

Эти отметки показаны на бланке для записи программы на КОБОЛе, приведенном на рис. 1.10. Место справа от отметки R не является частью строки ввода в КОБОЛ-компилятор и может использоваться с различными целями, как, например, для идентификации строк. Расстояние от отметки В до отметки R определяется каждой конкретной реализацией языка КОБОЛ, в большинстве случаев отметка R располагается между семьдесят второй и семьдесят третьей позициями литер, хотя возможны и другие варианты.

Отметки ограничивают поля, которые имеют определенное значение. Эти поля называются:

поле порядкового номера	ограничено отметками L и C
поле индикатора продолжения/комментария	ограничено отметками C и A
поле A	ограничено отметками A и B
поле B	ограничено отметками B и R

Поле порядкового номера может содержать шестизначный порядковый номер для идентификации строк исходной программы с тем, чтобы иметь возможность легко вернуть на место выпавшую карту. Поле порядкового номера используется не обязательно, но если значение внесено, то все строки должны иметь номера и номера должны быть расположены в возрастающем порядке.

В дополнение к статьям и предложениям имеется возможность включить комментарии для объяснения назначения различных частей программы. Они называются строками комментариев и обозначаются звездочкой (*) в поле индикатора продолжения/комментария. Строки комментариев могут появляться в любом месте исходной программы, кроме первой и последней строки. Комментарии могут записываться буквенно-цифровыми литерами в любом месте поля A или поля B. Эти заметки будут воспроизводиться в распечатке программы, но не будут транслироваться на язык машины и не повлияют на выполнение программы. Специальная форма строки комментария представлена литерой / (дробная черта) в поле индикатора. Она вызывает печать комментария на следующей странице документации.

ПОРЯДКОВЫЙ НОМЕР	ПРОДОЛЖЕНИЕ		30	40	50	60	72
	А	Б ТЕКСТ					
1	6 7 8	11 12					
	ОТМЕТКА "L"						
	ОТМЕТКА "C"						
	ОТМЕТКА "A"						
	ОТМЕТКА "B"						
	ОТМЕТКА "R"						
	ПОЛЕ ПОРЯДКОВОГО НОМЕРА						
	ПОЛЕ ИНДИКАТОРА ПРОДОЛЖЕНИЯ/КОММЕНТАРИЯ						
	ПОЛЕ "A"						
	ПОЛЕ "B"						

Рис. 1.10. Бланк для записи КОБОЛ-программы с указанием полей, ограниченных отметками.

Если предложение или статья не помещаются в одной строке, они могут быть легко продолжены в поле В следующей строки, если в строке имеется естественный разрыв в виде пробела. Например:

```
GO  
TO  
SEARCH-PROCEDURES.
```

является вполне правильной записью и означает то же самое, что

```
GO TO SEARCH-PROCEDURES.
```

Поле индикатора используется для обозначения продолжения, если только предложение или статья должны быть продолжены на следующей строке без единого пробела. Дефис в поле индикатора строки указывает, что первая литера, отличная от пробела в поле В данной строки, является непосредственным приемником последней отличной от пробела литеры в поле В предыдущей строки. Например:

```
GO TO SEA  
RCH-PROCEDURES.
```

то же самое, что GO TO SEARCH-PROCEDURES. Исключение составляют случаи, когда предыдущая строка содержит нечисловой литерал без завершающих кавычек. В таких случаях первой отличной от пробела литерой строки продолжения должен быть знак кавычки и продолжение начинается первой же литерой независимо от того, является ли она пробелом или нет, непосредственно следующей за знаком кавычки. Например, фраза

```
VALUE IS "ABSDEFG".
```

может быть разбита на две строки, так

```
VALUE IS "ABC (буква С должна быть рядом с отметкой R)  
"DEFG".
```

Все пробелы в поле В в конце продолжаемой строки считаются частью литерала. Приведенный пример будет верен только в том случае, если "ABC" заканчивается у отметки R.

Если в поле индикатора строки дефис отсутствует, то за последней литерой предыдущей строки автоматически следует пробел, даже если компилятору необходимо добавлять фиктивную позицию. Таким образом, имеется возможность полностью использовать поле В до последней позиции, даже если последняя литера — это точка, за которой обычно должен следовать пробел.

Для удобства чтения программы разрешается использовать целую строку пробелов. Строка пробелов не может непосредственно предшествовать строке продолжения.

Обычно для записи программы используются поля А и В. Некоторые части программы должны начинаться в поле А, другие

должны начинаться в поле В; отдельные части могут начинаться как в том, так и в другом поле. Требования такого рода будут точно определены во время описания отдельных частей программы. Общее правило состоит в том, что старшие заголовки должны начинаться в поле А, а младшие заголовки, статьи и предложения должны начинаться в поле В.

Упражнения

1. Подготовьте пример КОБОЛ-программы, приведенной на рис. 1.4, для ввода в имеющуюся вычислительную машину. Такое упражнение поможет познакомиться с требованиями местного вычислительного центра.

2. Найдите в тексте примеры фраз, статей, операторов, предложений и параграфов.

3. Укажите, что из приведенного ниже является словами, литерами, неверными словами или зарезервированными словами:

- а. JOHN-SMITH
- б. IDENTIFICATION DIVISION
- в. 12345
- г. 12.
- д. -1
- е. -A
- ж. ANTIDISESTABLISHMENTARIANISM
- з. VARY
- и. A-2-3-4-5-6
- к. BOB-O-LINK
- л. SOURCE-COMPUTER
- м. IN/OUT

4. В следующих двух примерах строк с продолжениями запишите эти строки без продолжений:

Отметка	Отметки		Отметка		Отметка
L	C	A	B		R
		FD	OLD-MASTER-FILE		LABEL RE
				CORDS ARE	
			STANDARD.		
				MOVE "NAME AND	
				"ADDRESS" TO ITEM-G.	

1.3. Четыре раздела КОБОЛа

Исходная КОБОЛ-программа всегда состоит из четырех основных частей, называемых разделами. Эти четыре раздела должны присутствовать в каждой программе и должны следовать в определенном порядке: раздел идентификации, раздел оборудования, раздел данных и раздел процедур. Эти разделы существуют для выделения различных функций исходной программы, а также для большей ясности и простоты модификации программы. Раздел идентификации служит для определения программы: ее имени и происхождения. Раздел оборудования описывает вычислительную машину, на которой будет выполняться программа. Раздел данных описывает и именует данные, записи и файлы, которые необходимо обработать. Раздел процедур содержит предложения, которые будут управлять действиями машины.

Каждый раздел может содержать много различных статей и предложений (большие программы могут содержать порядка 5000 строк). Целью настоящей главы является постепенное знакомство с различными возможностями. Таким образом в ней будут описаны лишь минимальные требования к каждому разделу. В последующих главах будут описываться различные дополнительные возможности.

Каждый раздел начинается заголовком раздела, который состоит из слова, обозначающего название раздела, и зарезервированного слова **DIVISION (РАЗДЕЛ)** и оканчивается точкой. Эти заголовки должны начинаться в поле А стандартного формата и должны быть в строке одни. Когда нечто описывается как начинающееся в поле А, это значит, что первая литера должна быть пробита в позициях 8, 9, 10 или 11. Таким образом, фиксируется только начальная позиция. Если потребуется, литеры могут размещаться во всем поле В.

Раздел идентификации

Минимальным требованием к разделу идентификации является то, что он должен содержать параграф **PROGRAM-ID (ПРОГРАММА)**. Другие параграфы необязательны и будут описаны в последующих главах. Заголовок параграфа **PROGRAM-ID** должен начинаться в поле А и заканчиваться точкой. Этот параграф может содержать только одну статью — имя-программы. Имя-программы определяется пользователем и идентифицирует программу в целом. Имя-программы должно начинаться в поле В и заканчиваться точкой. Любая запись, начинающаяся в поле В, может начинаться в любом месте поля В от позиции 12 до отметки R. Статья, начинающаяся в поле В, может быть в той же строке, что и предыдущая запись, начинающаяся в поле А. Имя-программы должно начинаться буквенной литерой и быть не длиннее семи литер, чтобы обеспечить

совместимость с другими КОБОЛ-компиляторами. Во всех случаях это должно быть допустимое слово КОБОЛа. Оно не должно содержать специальные литеры или дефис. Примеры полных минимальных разделов идентификации:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.      TEST.

IDENTIFICATION DIVISION.
PROGRAM-ID.      JBM 1234.
```

Это, безусловно, наиболее простая часть языка.

Раздел оборудования

Раздел оборудования содержит две секции. Каждая секция имеет заголовок секции, который должен начинаться в поле А и заканчиваться точкой. CONFIGURATION SECTION (СЕКЦИЯ КОНФИГУРАЦИИ) содержит информацию о машине, на которой будет транслироваться КОБОЛ-программа (исходная машина), и о машине, на которой будет выполняться рабочая программа (рабочая машина). Чаще всего это одна и та же машина, т. е. одна машина используется для трансляции и выполнения программы. Многие КОБОЛ-компиляторы не в состоянии составить рабочую программу для вычислительных машин других марок. На деле имя машины рассматривается многими компиляторами как комментарий к компилятору могут составить программу только для одной-единственной машины. Тем не менее следующие статьи всегда необходимы:

```
CONFIGURATION SECTION.
SOURCE-COMPUTER.      имя-машины.
OBJECT-COMPUTER.      имя-машины.
```

Слова *имя-машины* напечатаны строчными буквами для указания того, что это нечто предлагаемое пользователем; заглавные буквы, например SOURCE-COMPUTER, показывают, что это слова из списка зарезервированных слов КОБОЛа. Этот принцип будет последовательно соблюдаться в данной книге при описании формата КОБОЛа.

Оба имени-машины должны начинаться в поле В и заканчиваться точкой. Имя-машины описывает машину и определяется ее производителем, но это должно быть допустимое слово КОБОЛа и только одно слово; внутри него не разрешаются пробелы. Имя-машины может быть такого типа:

```
IBM-360-H50
B-6500
MARK-1
```

Точное имя должно быть определено в вычислительном центре, в котором будет транслироваться КОБОЛ-программа. Примеры секции конфигурации раздела оборудования:

CONFIGURATION SECTION.

SOURCE-COMPUTER. IBM-360-H50.

OBJECT-COMPUTER. IBM-360-H50.

CONFIGURATION SECTION.

SOURCE-COMPUTER. B-5500.

OBJECT-COMPUTER. B-5500.

Вторая секция на этом уровне описания языка имеет один параграф, называемый FILE-CONTROL (УПРАВЛЕНИЕ-ФАЙЛАМИ). Заголовок-параграфа FILE-CONTROL должен начинаться в поле А и заканчиваться точкой. Параграф содержит одну или несколько статей SELECT (ДЛЯ), каждая из которых должна начинаться в поле В и заканчиваться точкой. Запомните, что за каждой точкой должен следовать хотя бы один пробел. Общая форма статьи следующая:

SELECT имя-файла ASSIGN TO имя-устройства.

Имя-файла относится к файлу, который будет описываться в программе ниже в разделе данных и будет использоваться программой для операций ввода или вывода. Обычно в программе обрабатываются много файлов и каждый должен иметь собственное имя-файла. Имя-файла используется только в КОБОЛ-программе и присваивается программистом. Именем-файла служит слово КОБОЛа (до 30 литер, включая возможные дефисы), но оно должно начинаться буквенной литерой. Имя-файла определяет файл, который является совокупностью записей, в рамках программы, и к этому имени возможно обращение в разделе процедур. Записи этого файла хранятся в каком-либо устройстве машины, например на устройстве магнитной ленты. Целью статьи SELECT является указание программе, где может быть найден файл с данным именем: местонахождение файла задается именем устройства. В следующей главе более подробно рассматривается оборудование машины и упоминаются некоторые из ее устройств. Имя-устройства — это имя некоторого устройства внешних данных, которое уникально для каждой установки и определяется вычислительным центром так же, как и имя-машины. Примеры статей SELECT (каждый пример для вычислительной установки различного типа):

SELECT INPUT-FILE-G ASSIGN TO UT-S-SYSIN.

SELECT TEST-EXAMPLE-123 ASSIGN TO CARD-READER.

SELECT REFERENCE-FILE ASSIGN TO TAPE.

SELECT A-X-B ASSIGN TO MAGNETIC-TAPE-UNIT.

SELECT ANSWER-FILE ASSIGN TO UT-2400-S-RESULTS.

Итак, были даны минимальные сведения о двух разделах, которые в программе имеют вид, подобный приведенному ниже:

IDENTIFICATION DIVISION.

PROGRAM-ID. имя-программы.

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER. имя-машины-1.

OBJECT-COMPUTER. имя-машины-2.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

SELECT имя-файла-1 ASSIGN TO имя-устройства-1.

SELECT имя-файла-2 ASSIGN TO имя-устройства-2.

Возможно наличие многих статей SELECT, и в каждом случае имена, обозначенные строчными буквами, должны выбираться программистом.

Раздел данных

Третьим разделом является раздел данных. Он имеет две секции: секцию файлов и секцию рабочей-памяти. Заголовки секции файлов и секции рабочей-памяти начинаются в поле А и заканчиваются точкой. Секция файлов описывает формат каждого типа записей, которые содержатся в файлах, перечисленных в параграфе FILE-CONTROL. Для каждого файла в секции файлов указывается также его описание (его имя, сведения о наличии или отсутствии меток и др.). Секция файлов, таким образом, состоит из заголовка и следующих за ним статей-описания-файлов и статей-описания-записей, например:

FILE SECTION

статья-описания-файла-1.

статья-описания-записи-1.

статья-описания-файла-2.

статья-описания-записи-2.

и т. д. Каждая статья-описания-файла состоит из заголовка FD (ОФ), начинающегося в поле А, и следующего за ним имени-файла, начинающегося в поле В, с последующей фразой LABEL (МЕТКИ), например:

FD OLD-MASTER-FILE LABEL RECORD IS STANDARD.

Фраза LABEL может иметь разный вид, слово STANDARD (СТАНДАРТНЫ) может быть заменено на OMITTED (ОПУЩЕНЫ). Каж-

дая статья-описания-записи состоит из числового заголовка 01, начинающегося в поле А, с последующим подходящим именем-записи, начинающимся в поле В, за которым следует описательная фраза, указывающая формат записи. Рассмотрение статьи-описания-записи требует некоторых деталей, оно будет отложено до следующей главы, но примером может служить следующая статья:

01 INPUT-RECORD PICTURE IS X(450).

Эта статья означает, что запись имеет имя INPUT-RECORD и состоит из 450 позиций литер. Каждая позиция литер может содержать любую литеру из буквенно-цифрового набора литер.

Секция рабочей-памяти описывает записи, которых нет во внешних файлах, а которые создаются в рабочей-памяти машины и используются в процессе выполнения программы. Статьи в секции рабочей-памяти полностью состоят из статей-описания-данных, которые имеют точно такую же структуру, как статьи-описания-записей, т. е. заголовок 01 в поле А с последующим именем и затем описательной фразой. Дальнейшее более детальное описание раздела данных приведено в следующей главе. Ниже приводится пример раздела данных:

DATA DIVISION.

FILE SECTION.

FD INPUT-FILE LABEL RECORD IS OMITTED.

01 INPUT-FILE-RECORD PICTURE IS X(80).

FD OUTPUT-FILE LABEL RECORD IS STANDARD.

01 OUTPUT-FILE-RECORD PICTURE IS X(132).

WORKING-STORAGE SECTION.

01 CONSTANT-VALUE-X PICTURE IS X(5) VALUE IS 153.0.

Фраза VALUE IS (ЗНАЧЕНИЕ) содержит числовой литерал, чье значение равно ста пятидесяти трем. Самая правая точка завершает статью-описания-данных и не является частью литерала.

Раздел процедур

Четвертый, заключительный раздел программы — раздел процедур. Это раздел программы, описывающий действия. Первые три раздела содержат описательные статьи, состоящие из фраз. Этот раздел содержит предложения, состоящие из операторов. Раздел начинается заголовком PROCEDURE DIVISION, начинающимся в поле А и заканчивающимся точкой. Раздел может содержать секции, но в простейшей форме они опускаются. Раздел всегда составляется из *параграфов*, которые являются поименованными по-

следовательностями предложений. В КОБОЛе один или более параграфов принято называть *процедурой*. Поэтому раздел называется разделом процедур.

Параграф состоит из заданного программистом имени-параграфа, начинающегося в поле А и заканчивающегося точкой, за которой следует один или более пробелов, а затем одно или более предложений. Каждое предложение должно быть включено в некоторый параграф. Предложение должно начинаться в поле В и заканчиваться точкой. Предложение — это один или несколько операторов, за которыми следует точка. Чаще всего предложение состоит из одного оператора, заканчивающегося точкой, но иногда используются составные, или сложные, предложения. Оператор — это комбинация глаголов и фраз, и он всегда начинается с глагола КОБОЛа. Почти вся эта книга посвящена обсуждению операторов и способов их использования, так что сейчас мы не будем рассматривать вопрос о формировании операторов. Выполнение операторов в разделе процедур начинается с выполнения первого записанного оператора и продолжается в порядке их следования в программе, за исключением случаев, когда правила предписывают другой порядок, как в случае оператора GO TO. Примером раздела процедур может быть:

PROCEDURE DIVISION.

PARAGRAPH-ONE.

MOVE STARTING-NUMBER TO FIRST-DATA-ITEM.

WRITE OUTPUT-RECORD-X.

ADD INCREMENT-VALUE TO STARTING-NUMBER.

IF STARTING-NUMBER IS LESS THAN 100 GO TO
PARAGRAPH-ONE.

STOP RUN.

1.4. Пример программы

КОБОЛ-программа состоит из четырех разделов. Раздел идентификации прост и требует лишь задания короткого идентификационного имени. Раздел оборудования скорее всего будет различным для каждой вычислительной машины; именно поэтому этот раздел и необходим. Если КОБОЛ-программу надо использовать на другой машине, раздел оборудования можно изменить и программу транслировать на новую машину. Раздел данных имеет две секции; эти две секции описывают формат всех данных, которые будут обрабатываться в разделе процедур. Дальнейшее объяснение статей-описания-записей и статей-описания-данных будет дано позднее. А сейчас приведем образец программы.

000010 IDENTIFICATION DIVISION.
000020 PROGRAM-ID. FIRST.
000030 ENVIRONMENT DIVISION.
000040 CONFIGURATION SECTION.
000050 SOURCE-COMPUTER. CDC-6600.
000060 OBJECT-COMPUTER. CDC-6600.
000070 * ВСЕ, ПЕРЕЧИСЛЕННОЕ ВЫШЕ, НЕОБХОДИМО.
000080 * НИКАКИЕ СОКРАЩЕНИЯ НЕ ДОПУСКАЮТСЯ.
000090 INPUT-OUTPUT SECTION.
000100 FILE-CONTROL.
000110 SELECT OLD-FILE ASSIGN TO CARD-READER.
000120 SELECT NEW-FILE ASSIGN TO PRINTER.
000130 * ИМЕНА-ФАЙЛОВ ВЫБИРАЕТ ПОЛЬЗОВАТЕЛЬ.
000140 * ИМЕНА-УСТРОЙСТВ НАЗНАЧАЮТСЯ В ВЦ.
000150 DATA DIVISION.
000160 FILE SECTION.
000170 FD OLD-FILE LABEL RECORD IS STANDARD.
000180 * ЕСЛИ НЕТ СПЕЦИАЛЬНЫХ ТРЕБОВАНИЙ,
000190 * ИСПОЛЬЗОВАТЬ ВАРИАНТ STANDARD.
000200 01 OLD-FILE-RECORD PICTURE IS X(80).
000210 * ИМЯ-ЗАПИСИ НЕ ДОЛЖНО СОВПАДАТЬ С
000220 * ИМЕНЕМ-ФАЙЛА.
000230 FD NEW-FILE
000240 LABEL RECORD IS STANDARD.
000250 * ОБРАТИТЕ ВНИМАНИЕ, ЧТО ФРАЗЫ МОГУТ
000260 * РАСПОЛАГАТЬСЯ В НЕСКОЛЬКИХ СТРОКАХ.
000270 01 RECORD-IN-THE-NEW-FILE.
000280 02 RECORD-SPACE-X PICTURE IS X(80).
000290 02 COUNT-OF-RECORDS PICTURE IS 9(5).
000300 * ЭТА СТАТЬЯ-ОПИСАНИЯ-ЗАПИСИ ОПИСЫВАЕТ
000310 * ЗАПИСЬ ИЗ ДВУХ ЭЛЕМЕНТАРНЫХ ДАННЫХ.
000320 WORKING-STORAGE SECTION.
000330 01 RECORD-COUNT PICTURE IS 9(5) VALUE IS 1.
000340 * 9 В ШАБЛОНЕ ОЗНАЧАЕТ, ЧТО ДАННОЕ МОЖЕТ
000350 * СОСТОЯТЬ ТОЛЬКО ИЗ ДЕСЯТИЧНЫХ ЦИФР.
000360 PROCEDURE DIVISION.
000370 FIRST-STEP.

```
000380      OPEN INPUT OLD-FILE.
000390      OPEN OUTPUT NEW-FILE.
000400 *   ВСЕ ФАЙЛЫ ДОЛЖНЫ БЫТЬ ОТКРЫТЫ.
000410      NEXT-X.
000420      READ OLD-FILE RECORD AT END STOP RUN.
000430      MOVE OLD-FILE TO RECORD-SPACE-X.
000440 *   СЧИТАННЫЕ 80 ЛИТЕР ПОМЕЩАЮТСЯ
000450 *   В ОБЛАСТЬ ПАМЯТИ RECORD-SPACE-X.
000460      MOVE RECORD-COUNT TO COUNT-OF-RECORDS.
000470      WRITE RECORD-IN-THE-NEW-FILE.
000480      ADD 1 TO RECORD-COUNT.
000490 *   ДЛЯ ЗАДАНИЯ КОНЦА ПРОГРАММЫ НЕ
000500 *   ТРЕБУЕТСЯ НИКАКОГО СПЕЦИАЛЬНОГО
000510 *   УКАЗАТЕЛЯ, НО СТРОКА КОММЕНТАРИЯ
000520 *   НЕ МОЖЕТ БЫТЬ ПОСЛЕДНЕЙ СТРОКОЙ
000530 *   ПРОГРАММЫ.
000540      GO TO NEXT-X.
```

Упражнения

1. Каково назначение PROGRAM-ID?
2. Каковы подразделения параграфа?
3. Какова максимальная длина слова?
4. Опишите два типа литералов.
5. Назовите четыре раздела КОБОЛ-программы.
6. Напишите четыре различных зарезервированных слова КОБОЛа.
7. Составьте имя-данного, имя-параграфа, имя-файла и имя-записи.
8. Где в строке КОБОЛ-программы находятся отметки A и B?
9. Какова разница между секциями FILE SECTION и WORKING-STORAGE SECTION?
10. Напишите предложение.

1.5. Формальное описание языка

Обучая грамматике естественного языка, такого, как английский, преподаватели часто сталкиваются с вопросами студентов: «В чем же отличие?». Действительно, если употребить «can» вместо

«may» или «shall» вместо «will», то в разговорном английском это не будет грубой ошибкой. Но в языке команд вычислительной машины требуется абсолютная точность. Если в разговоре вы допустили языковую неточность, то можно ожидать, что собеседник все равно поймет вас правильно. Иначе обстоит дело при общении с машиной. Она реагирует на команду SEARCH REFERENCE-TABLE по-одному, а на команду SEARCH ALL REFERENCE-TABLE совершенно по-другому. При определении структуры и смысла статей и предложений КОБОЛа необходимо быть пунктуальными с тем, чтобы программист точно представлял себе результаты их использования. Употребление специальной нотации, называемой общим форматом, существенно облегчает описание структуры языковых конструкций. Общие форматы описывают специфическую организацию элементов фраз и операторов, фиксируют порядок слов и не обязательных альтернатив, а также показывают, куда должны помещаться слова, выбираемые программистом.

Общие форматы состоят из слов, написанных заглавными буквами, слов, написанных строчными буквами, фигурных и квадратных скобок и некоторых специальных символов. Общий формат по сути является еще одним языком, *мета-языком*, который используется для формального описания языка КОБОЛ. В этом мета-языке слова, написанные заглавными буквами, являются зарезервированными словами КОБОЛа и должны записываться точно, как они определены. Подчеркнутые слова, написанные заглавными буквами, являются ключевыми словами, и они обязательны. Неподчеркнутые слова, написанные заглавными буквами, не обязательны и могут быть не включены. Они называются вспомогательными словами и не обязательны, но программистам, особенно начинающим, настоятельно рекомендуется использовать их. Слова, написанные строчными буквами, заменяются одним или несколькими словами, составляемыми в соответствии с различными правилами. Например:

SELECT имя-файла ASSIGN TO имя-устройства.

является общим форматом, описывающим статью SELECT в параграфе FILE-CONTROL. Слова SELECT и ASSIGN — это ключевые, обязательные слова, в то время как TO необязательное, вспомогательное слово. Два слова, написанные строчными буквами, выбираются программистом при составлении конкретной статьи. Таким образом, общий формат статьи SELECT показывает, что приведенная ниже статья является правильной статьей SELECT:

SELECT MY-OWN-FILE-NAME ASSIGN TO SOME-DEVICE.

За каждой точкой, показанной в общем формате, должен следовать один или более пробелов.

Отсутствие линии, подчеркивающей одиночное слово, написанное заглавными буквами, указывает на то, что это слово не обязательно; если часть общего формата заключена в квадратные скобки [], то она целиком может быть включена или опущена по усмотрению пользователя. Эта часть не обязательна, но если она включена, то она может изменить смысл фразы или оператора. Например:

WRITE имя-записи [FROM имя-данного]

устанавливает, что фраза «FROM имя-данного» может быть включена или нет в оператор WRITE. Если эта фраза будет включена, то слово FROM будет ключевым словом, которое является обязательным. Добавление этой фразы меняет действия. Сравним два оператора:

WRITE RECORD-IN-THE-NEW-FILE

и

WRITE RECORD-IN-THE-NEW-FILE
FROM OLD-RECORD

В первом случае будет выдано (записано) текущее значение указанной записи, а во втором — это значение будет перед записью заменено на значение записи OLD-RECORD.

Если имеются различные возможности выбора, но выбрана *должна* быть только одна из них, это указывается путем расположения этих возможностей друг под другом и заключения их в фигурные скобки { }. Например:

LABEL { RECORD IS } { STANDARD }
 { RECORDS ARE } { OMITTED }

означает, что любая из фраз, перечисленных ниже, является правильной фразой КОБОЛа:

LABEL RECORD IS STANDARD
LABEL RECORD IS OMITTED
LABEL RECORD STANDARD
LABEL RECORDS ARE STANDARD
LABEL RECORDS OMITTED

И здесь бы мы рекомендовали включать все вспомогательные слова при написании программы.

Последней специальной нотацией общего формата является многоточие (три точки...), которое указывает, что по усмотрению про-

граммиста возможны повторения. Многоточие всегда следует за квадратными или фигурными скобками, означая, что конструкция, заключенная в скобки, может быть повторена неограниченное число раз. Например:

FILE-CONTROL. {SELECT имя-файла
ASSIGN TO имя-устройства.}...

указывает, что статья SELECT в параграфе FILE-CONTROL один раз должна быть использована обязательно, но может повторяться столько раз, сколько потребуется. Заметьте, что повторение не означает переписывания одной и той же фразы несколько раз, а означает повторение различных допустимых вариантов. Например:

FILE-CONTROL.
SELECT A-FILE ASSIGN TO TAPE.
SELECT B-FILE ASSIGN TO DISK.

В нотации общего формата нет указания на то, в каком поле должны начинаться отдельные части, и нет указания на смысл окончательной конструкции КОБОЛа. Такая информация должна быть добавлена в текст. Целью общего формата является описание формата возможно более коротко и точно. Потребовалось бы довольно большое словесное описание того, как сконструировать предложение USE (ИСПОЛЬЗОВАТЬ). Ниже приводится описание этого предложения через общий формат:

<u>USE</u>	{	<u>BEFORE</u>		<u>STANDARD</u>	{	<u>BEGINNING</u>		<u>LABEL PROCEDURE ON</u>
		<u>AFTER</u>				<u>ENDING</u>		
		{	имя-файла	...	}			
			<u>INPUT</u>					
			<u>OUTPUT</u>					

Согласно этому описанию, следующие предложения верны:

USE BEFORE STANDARD ENDING LABEL PROCEDURE ON INPUT.
USE AFTER STANDARD ENDING LABEL PROCEDURE OUTPUT.
USE BEFORE STANDARD BEGINNING LABEL PROCEDURE ON A-FILE B-FILE C-FILE.

Пример

Приводится словесное описание оператора КОБОЛа. Напишите общий формат, определяющий этот оператор.

Оператор поиска представляет собой глагол *search*, за которым следует имя некоторой таблицы, две необязательные и одна обязательная фраза. Первая фраза представляет собой обязательное слово *varying* и следующую за ним одну из трех возможностей: имя-индекса, имя-индексного-данного или имя-данного. Может быть указана только одна. Названные возможности взаимно исключают друг друга. Вторая фраза не является обязательной, это фраза «*at end*». Она имеет обязательные слова «*at end*», за которыми следует любой оператор КОБОЛа. Слово *at* необязательно. Конечная (также необязательная) фраза имеет необходимое слово *when*, за которым следует так называемое «условие», а затем — либо повелительные операторы, либо специальные слова *next sentence*. Самая последняя фраза может повторяться столько раз, сколько потребуется.

Решение

Сказанное выше демонстрирует, что английский язык не очень хорош, когда требуется точное описание предмета. В общем формате оператор поиска будет определен следующим образом:

SEARCH имя-таблицы VARYING $\left\{ \begin{array}{l} \text{имя-индекса} \\ \text{имя-индексного-данного} \\ \text{имя-данного} \end{array} \right\}$
 [AT END {повелительный-оператор} ...]
 [WHEN условие $\left\{ \begin{array}{l} \text{[повелительный-оператор]} \dots \\ \text{NEXT SENTENCE} \end{array} \right\} \dots] \dots$

Заметьте, что в этой спецификации не делается попытки объяснить, что такое на самом деле имя-таблицы, имя-индексного-данного или условие и т. п. Общий формат имеет дело с синтаксисом, а не с семантикой (со смыслом конструкций) языка.

Упражнение

Напишите допустимый оператор КОБОЛа для каждого из следующих общих форматов:

$$\text{ADD} \left\{ \begin{array}{l} \text{имя-данного-1} \\ \text{литерал} \end{array} \right\} \text{ TO имя-данного-2 [ROUNDED]}$$

$$\text{MULTIPLY} \left\{ \begin{array}{l} \text{имя-данного-1} \\ \text{литерал-1} \end{array} \right\} \text{ BY } \left\{ \begin{array}{l} \text{имя-данного-2} \\ \text{литерал-2} \end{array} \right\} \\ \text{GIVING имя-данного-3 [ROUNDED]}$$

$$\text{I-O-CONTROL} \left[\text{RERUN} \left[\text{ON} \left\{ \begin{array}{l} \text{имя-файла-1} \\ \text{имя-устройства} \end{array} \right\} \right] \right. \\ \left. \text{EVERY} \left\{ \begin{array}{l} \text{литерал-1 RECORDS OF имя-файла-2} \\ \text{литерал-2 CLOCK-UNITS} \end{array} \right\} \right] \dots$$

1.6. Глоссарий

Глагол. Зарезервированное слово, обозначающее действие, которое нужно произвести рабочей КОБОЛ-программе. Примерами глаголов КОБОЛа являются READ, WRITE, MOVE, GO.

Заголовок параграфа. Слово КОБОЛа, за которым непосредственно следует точка с последующими одним или более пробелами, которое идентифицирует и предваряет все статьи в разделах идентификации и оборудования и все последовательности предложений в разделе процедур. Всегда начинается в поле А.

Заголовок раздела. Слово КОБОЛа, указывающее на начало отдельного раздела, за которым следует зарезервированное слово DIVISION и точка. Заголовки разделов всегда начинаются в поле А формата представления.

Заголовок секции. Комбинация слов, второе из которых является зарезервированным словом SECTION, указывающим начало каждой секции в разделе оборудования, данных и процедур. Всегда начинается в поле А и оканчивается точкой. Заголовками секций являются:

В разделе оборудования:

CONFIGURATION SECTION
INPUT-OUTPUT SECTION

В разделе данных:

FILE SECTION
WORKING-STORAGE SECTION

В разделе процедур:

Определяемое пользователем имя-секции с последующим зарезервированным словом SECTION, заканчивающимся точкой.

Имя-данного. Слово КОБОЛа, определенное программистом, содержащее по крайней мере одну буквенную литеру, обычно начальную, и именующее статью данных в разделе данных.

Имя-программы. Слово, идентифицирующее исходную КОБОЛ-программу.

Имя-устройства. Слово, определяющее тип устройства (его особые физические свойства). Оно определяется создателем компилятора КОБОЛа и при использовании становится зарезервированным словом.

Исходная КОБОЛ-программа. Форма представления алгоритма для решения задачи обработки данных, использующая язык, формат и синтаксис КОБОЛа.

Литера. Основная неделимая единица языка КОБОЛ.

Литера буквенная. Литера, являющаяся одной из прописных букв А, В, С, ..., Х, Y, Z, а также пробел.

Литера буквенно-цифровая. Любая литера из набора литер машины; количество может быть различным для различных машин и может достигать до 256.

Литера КОБОЛа. Литера, принадлежащая набору из 71 литеры, состоящему из буквенных, цифровых и специальных литер.

Литера специальная. Литера, принадлежащая следующему набору из 14 литер: + - * / , ; . " = \$ () > <.

Литера цифровая. Литера, являющаяся одной из десяти цифр (0, 1, 2, ..., 8, 9).

Литерал. Строка литер, значение которой определяется литерами, составляющими эту строку. Имеется два типа литералов: числовые и нечисловые. Числовые литералы могут содержать только цифровые литеры, одну десятичную точку и один знак. Нечисловые литералы заключаются в кавычки.

Оператор. Синтаксически правильная комбинация слов и символов, начинающаяся глаголом и записанная в разделе процедур.

Предложение. Последовательность из одного или нескольких операторов, последний из которых заканчивается точкой с последующим пробелом.

Рабочая программа. Совокупность машинных команд, которые являются результатом трансляции исходной КОБОЛ-программы.

Раздел. Одна или более секций или параграфов, которые формируются или составляются в соответствии со специальными правилами. КОБОЛ-программа всегда состоит из четырех разделов: раздела идентификации, раздела оборудования, раздела данных и раздела процедур.

Слово. Последовательность, состоящая не более чем из 30 литер. Каждая литера принадлежит набору буквенных литер (за исключением пробела), цифровых литер и единственной специальной литеры дефиса (-). Дефис не может быть первой или последней литерой слова.

Статья. Любой набор последовательных фраз описания, заканчивающихся точкой (с одним или более пробелами, следующими за ней) и записанных в трех первых разделах программы.

УПРАВЛЕНИЕ-ФАЙЛАМИ(FILE-CONTROL). Имя параграфа раздела оборудования, в котором для файлов данных конкретной исходной программы объявляются имена и назначаются устройства той установки, на которой будет работать программа.

Формат общий. Спецификация типа и порядка элементов, образующих операторы и фразы КОБОЛа. Спецификация использует слова, записанные заглавными буквами, строчными буквами, специальные символы и специальные литеры.

Формат представления. Формат, который обеспечивает стандартный способ описания исходной КОБОЛ-программы с помощью строк, отметок и полей.

Фраза. Упорядоченная последовательность слов КОБОЛа, назначением которой является определение свойства статьи.

Список зарезервированных слов КОБОЛа,
не включающий слова, которые могут быть
определены в конкретной реализации

ACCEPT (ПРИНЯТЬ)
ACCESS (ДОСТУП)
ADD (СЛОЖИТЬ)
ADVANCING (ПРОДВИЖЕНИЯ)
AFTER (ЗАТЕМ, ПОСЛЕ)
ALL (ВСЕ, ОСОБО, ВСЕХ, ВСЕМИ)
ALPHABETIC (БУКВЕННОЕ)
ALTER (ИЗМЕНИТЬ)
ALTERNATE (ДОПОЛНИТЕЛЬНЫЙ)
AND (И)
ARE
AREA (ОБЛАСТЬ)
AREAS (ОБЛАСТЕЙ)
ASCENDING (ПО ВОЗРАСТАНИЮ)
ASSIGN TO (НАЗНАЧИТЬ)
AT (ОТ, В)
AUTHOR (АВТОР)
BEFORE (ДО)
BLANC (ПРОБЕЛ)
BLOCK (БЛОКЕ)
BOTTOM (НИЖНЕЕ ПОЛЕ)
BY (НА)
CALL (ВЫЗВАТЬ)
CANCEL (ОСВОБОДИТЬ)
CD (OK)
CF (УК)
CH (УЗ)
CHARACTER (ЛИТЕРА)
CHARACTERS (ЛИТЕР, ЛИТЕРЫ)
CLOCK-UNITS (ЕДИНИЦ-ВРЕМЕНИ)
CLOSE (ЗАКРЫТЬ)
COBOL (КОБОЛ)
CODE (С КОДОМ)
CODE-SET (АЛФАВИТ)
COLUMN (СТОЛБЦА)

СОММА (ЗАПЯТАЯ)
COMMUNICATION (КОММУНИКАЦИЙ)
COMP (ВЫЧ)
COMPUTATIONAL (ВЫЧИСЛЕНИЙ)
COMPUTE (ВЫЧИСЛИТЬ)
CONFIGURATION (КОНФИГУРАЦИИ)
CONTAINES (В)
CONTROL (УПРАВЛЕНИЕ ПО, УПРАВЛЯЕМЫЙ,
УПРАВЛЯЕМАЯ)
CONTROLS (УПРАВЛЕНИЕ ПО)
COPY (КОПИРОВАТЬ)
CORR (СООТВ)
CORRESPONDING (СООТВЕТСТВЕННО)
COUNT (ЧИСЛО, СЧЕТ)
CURRENCY (ВАЛЮТНЫЙ)
DATA (ДАНЫХ)
DATE (ДАТА, ДАТУ)
DATE-COMPILED (ДАТА-КОМПИЛЯЦИИ)
DATE-WRITTEN (ДАТА-НАПИСАНИЯ)
DAY (ДЕНЬ)
DE (ФР)
DEBUG-CONTENTS (ЗНАЧЕНИЕ-ОТЛАДКИ)
DEBUG-ITEM (ДАНЫЕ-ОТЛАДКИ)
DEBUG-LINE (СТРОКА-ОТЛАДКИ)
DEBUG-SUB-1 (ИНДЕКС-ОТЛАДКИ-1)
DEBUG-SUB-2 (ИНДЕКС-ОТЛАДКИ-2)
DEBUG-SUB-3 (ИНДЕКС-ОТЛАДКИ-3)
DEBUG-NAME (ИМЯ-ОТЛАДКИ)
DEBUGGING (ОТЛАДКИ)
DECIMAL-POINT (ДЕСЯТИЧНАЯ-ТОЧКА)
DECLARATIVES (ДЕКЛАРАТИВЫ, ДЕКЛАРАТИВ)
DELETE (УДАЛИТЬ)
DELIMITED (ОГРАНИЧИВАЯСЬ)
DELIMITER (ОГРАНИЧИТЕЛЬ)
DEPENDING (В ЗАВИСИМОСТИ)
DEPTH
DESCENDING (ПО УБЫВАНИЮ)
DESTINATION (АДРЕСАТОВ, АДРЕСАТ)

DETAIL (ФРАГМЕНТ)
DISABLE (ЗАПРЕТИТЬ)
DISPLAY (ВЫДАЧИ, ВЫДАТЬ)
DIVIDE (РАЗДЕЛИТЬ)
DIVISION (РАЗДЕЛ)
DOWN (ВЫЧИТАЯ)
DUPLICATES (ДУБЛИРОВАНИЕМ)
DYNAMIC (ДИНАМИЧЕСКИЙ)
EGI (ИКГ)
ELSE (ИНАЧЕ)
EMI (ИКЩ)
ENABLE (РАЗРЕШИТЬ)
END (КОНЦА, КОНЕЦ, КОНЦЕ)
END-OF-PAGE (КОНЦЕ СТРАНИЦЫ)
ENTER (ВОЙТИ В)
ENVIRONMENT (ОБОРУДОВАНИЯ)
EOP (КОНЦЕ СТРАНИЦЫ)
EQUAL (РАВНО, РАВЕН)
ERROR (ОШИБКИ)
ESI (ИКС)
EVERY (КАЖДЫЙ, КАЖДЫЕ, КАЖДОЕ)
EXERTION (ОШИБКИ)
EXIT (ВЫЙТИ)
EXTEND (ДОПОЛНЯЕМЫХ)
FD (ОФ)
FILE (ФАЙЛА, ФАЙЛОВ)
FILE-CONTROL (УПРАВЛЕНИЕ-ФАЙЛАМИ)
FILLER (ЗАПОЛНИТЕЛЬ, ЗАП)
FINAL (КОНЦУ)
FIRST (ПЕРВЫЙ)
FOOTING (КОНЦОВКА)
FOR (ДЛЯ, С)
FROM (ОТ, ИЗ ПОЛЯ, С)
GENERATE (ГЕНЕРИРОВАТЬ)
GIVING (ПОЛУЧАЯ)
GO (ПЕРЕЙТИ)
GREATER (БОЛЬШЕ)
GROUP (ГРУППА, ГРУППУ)

HEADING (ЗАГОЛОВОК)
HIGH-VALUE (НАИБОЛЬШЕЕ-ЗНАЧЕНИЕ)
HIGH-VALUES (НАИБОЛЬШИЕ-ЗНАЧЕНИЯ)
I-O (ВХОДНЫХ-ВЫХОДНЫХ)
I-O-CONTROL (УПРАВЛЕНИЕ-ВВОДОМ-ВЫВОДОМ)
IDENTIFICATION (ИДЕНТИФИКАЦИИ)
IF (ЕСЛИ)
IN (В, ИЗ)
INDEX (ИНДЕКСА)
INDEXED (ИНДЕКСНАЯ, ИНДЕКСИРУЕТСЯ)
INDICATE (ОПРЕДЕЛЯЕТ)
INITIAL (НАЧАЛЬНОГО)
INITIATE (НАЧАТЬ)
INPUT (ВВОД, ВВОДА, ВХОДНЫХ)
INPUT-OUTPUT (ВВОДА-ВЫВОДА)
INSPECT (ПРОСМОТРЕТЬ)
INSTALLATION (ПРЕДПРИЯТИЕ)
INTO (В)
INVALID (ПРИ ОШИБКЕ)
IS (ЕСТЬ)
JUST (СДВИНУТО)
JUSTIFIED (СДВИНУТО)
KEY (КЛЮЧ, КЛЮЧА)
LABEL (МЕТКИ)
LAST (ПОСЛЕДНИЙ)
LEADING (ПЕРВЫЙ, ВЕДУЩИЕ)
LEFT (ВЛЕВО)
LENGTH (ДЛИНА)
LESS (МЕНЬШЕ)
LIMIT (РАЗМЕР)
LIMITS (РАЗМЕР)
LINAGE (ВЕРСТКА)
LINAGE-COUNTER (СЧЕТЧИК-ВЕРСТКИ)
LINE (СТРОК, СТРОКИ)
LINE-COUNTER (СЧЕТЧИК-СТРОК)
LINES (СТРОК)
LINKAGE (СВЯЗИ)
LOCK (ЗАМКОМ)

LOW-VALUE (НАИМЕНЬШЕЕ-ЗНАЧЕНИЕ)
LOW-VALUES (НАИМЕНЬШИЕ-ЗНАЧЕНИЯ)
MEMORY (ПАМЯТИ)
MERGE (СЛИТЬ)
MESSAGE (СООБЩЕНИЕ, СООБЩЕНИЙ, СООБЩЕНИЯ)
MODE (РЕЖИМЕ)
MODULES (МОДУЛЕЙ)
MOVE (ПОМЕСТИТЬ)
MULTIPLE, MULTIPLE FILE TAPE CONTAINES (НА ОДНОЙ
КАТУШКЕ)
MULTIPLY (УМНОЖИТЬ)
NEGATIVE (ОТРИЦАТЕЛЬНО)
NEXT (СЛЕДУЮЩАЯ, СЛЕДУЮЩЕЕ, СЛЕДУЮЩЕЙ,
СЛЕДУЮЩУЮ)
NO (НЕТ)
NOT (НЕ)
NUMBER (НОМЕР)
NUMERIC (ЧИСЛОВОЕ)
OBJECT-COMPUTER (РАБОЧАЯ-МАШИНА)
OCCURS (ПОВТОРЯЕТСЯ)
OF (НА, ИЗ)
OFF, OFF STATUS (ВЫКЛЮЧЕНО)
OMITTED (ОПУЩЕНЫ)
ON (НА, ПО, ПРИ, ДЛЯ, ОТ)
OPEN (ОТКРЫТЬ)
OPTIONAL (НЕОБЯЗАТЕЛЬНО)
OR (ИЛИ)
ORGANIZATION (ОРГАНИЗАЦИЯ)
OUTPUT (ВЫВОД, ВЫВОДА, ВЫХОДНЫХ)
OVERFLOW (ПЕРЕПОЛНЕНИИ)
PAGE (СТРАНИЦЕ, СТРАНИЦЫ)
PAGE-COUNTER (СЧЕТЧИК-СТРАНИЦ)
PERFORM (ВЫПОЛНИТЬ)
PF (КС)
PH (ЗС)
PIC (Ш)
PICTURE (ШАБЛОН)
PLUS (ПЛЮС)

POINTER (УКАЗАТЕЛЬ)
POSITIVE (ПОЛОЖИТЕЛЬНО)
PRINTING (ПЕЧАТЬ)
PROCEDURE (ПРОЦЕДУР, ПРОЦЕДУРА, ПРОЦЕДУРЫ)
PROCEDURES (ПРОЦЕДУРАХ)
PROCEED TO (ДЛЯ ПЕРЕХОДА)
PROGRAM (ПРОГРАММНЫЙ, ИЗ ПРОГРАММЫ)
PROGRAM-ID (ПРОГРАММА)
QUEUE (ОЧЕРЕДЬ)
QUOTE (КАВЫЧКА)
QUOTES (КАВЫЧКИ)
RANDOM (ПРОИЗВОЛЬНЫЙ)
RD (OO)
READ (ЧИТАТЬ)
RECEIVE (ПОЛУЧИТЬ)
RECORD (ЗАПИСИ, ЗАПИСЬ)
RECORDS (ЗАПИСЕЙ, ЗАПИСИ)
REDEFINES (ПЕРЕОПРЕДЕЛЯЕТ)
REEL (КАТУШКИ, КАТУШКУ)
REFERENCES (ССЫЛКАХ)
RELATIVE (ОТНОСИТЕЛЬНАЯ, ОТНОСИТЕЛЬНЫЙ)
RELEASE (ПЕРЕДАТЬ)
REMAINDER (ОСТАТОК)
REMOVAL (УДАЛЕНИЕМ)
RENAMES (ПЕРЕИМЕНОВЫВАЕТ)
REPLACING (ЗАМЕНЯЯ)
REPORT (ОТЧЕТ, ОТЧЕТА, ОТЧЕТОВ)
REPORTING (ВЫДАЧИ)
REPORTS (ОТЧЕТЫ)
RERUN (ПЕРЕПРОГОН)
RESERVE (РЕЗЕРВИРОВАТЬ)
RESET (СБРОСИТЬ)
RETURN (ВЕРНУТЬ)
REVERSED (РЕВЕРСНО)
REWIND (ПЕРЕМОТКИ)
REWRITE (ОБНОВИТЬ)
RF (KO)
RH (ZO)

RIGHT (ВПРАВО)
ROUNDED (ОКРУГЛЯЯ)
RUN (РАБОТУ)
SAME (ОБЩАЯ)
SD (ОС)
SEARCH (ИСКАТЬ)
SECTION (СЕКЦИЯ)
SECURITY (ПОЛНОМОЧИЯ)
SEGMENT (СЕГМЕНТ)
SEGMENT-LIMIT (ГРАНИЦА СЕГМЕНТОВ)
SELECT (ДЛЯ)
SEND (ПОСЛАТЬ)
SENTENCE (ПРЕДЛОЖЕНИЕ)
SEPARATE (ОТДЕЛЬНО)
SEQUENTIAL (ПОСЛЕДОВАТЕЛЬНАЯ,
ПОСЛЕДОВАТЕЛЬНЫЙ)
SET (УСТАНОВИТЬ)
SIGN (ЗНАК)
SIZE (РАЗМЕР, РАЗМЕРОМ)
SORT (СОРТ, СОРТИРОВАТЬ, СОРТИРОВКИ)
SORT-MERGE (СОРТИРОВКИ-СЛИЯНИЯ)
SOURCE (ИСТОЧНИК)
SOURCE-COMPUTER (ИСХОДНАЯ-МАШИНА)
SPACE (ПРОБЕЛ)
SPACES (ПРОБЕЛЫ)
SPECIAL-NAMES (СПЕЦИАЛЬНЫЕ-ИМЕНА)
STANDARD (СТАНДАРТ, СТАНДАРТНОЙ, СТАНДАРТНЫ)
START (ПОДВЕСТИ ЗАПИСЬ)
STATUS (СОСТОЯНИЕ, СОСТОЯНИЯ)
 ON STATUS (ВКЛЮЧЕНО)
 OFF STATUS (ВЫКЛЮЧЕНО)
STOP (ОСТАНОВИТЬ)
STRING (СОБРАТЬ)
SUB-QUEUE-1 (ПОДОЧЕРЕДЬ-1)
SUB-QUEUE-2 (ПОДОЧЕРЕДЬ-2)
SUB-QUEUE-3 (ПОДОЧЕРЕДЬ-3)
SUBTRACT (ОТНЯТЬ)
SUM (СУММА)

SUPPRESS (ПОДАВИТЬ)
SYMBOLIC (СИМВОЛИЧЕСКАЯ)
SYNC (ВЫДЕЛЕНО)
SYNCHRONIZED (ВЫДЕЛЕНО)
TABLE (ТАБЛИЦА)
TALLYING (СЧИТАЯ)
TAPE
TERMINAL (С ТЕРМИНАЛА)
TERMINATE (ЗАКОНЧИТЬ)
TEXT (ТЕКСТА)
THAN
THROUGH (ПО)
THRU (ПО)
TIME (ВРЕМЯ)
TIMES (РАЗ)
TO (ДО, К, В, С, НА)
TOP (ВЕРХНЕЕ ПОЛЕ)
TRAILING (ПОСЛЕДНИЙ)
TYPE (ТИП)
UNIT (ТОМ, ТОМА)
UNSTRING (РАЗОБРАТЬ)
UNTIL (ДО)
UP (ПРИБАВЛЯЯ)
UPON (ДЛЯ, НА)
USAGE (ДЛЯ)
USE (ИСПОЛЬЗОВАТЬ)
USING (ИСПОЛЬЗУЯ)
VALUE (ЗНАЧЕНИЕ, ЗНАЧ)
VALUES (ЗНАЧЕНИЕ, ЗНАЧ)
VARYING (МЕНЯЯ)
WHEN (КОГДА)
WITH (В, С)
WITH NO (БЕЗ)
WORDS (СЛОВ)
WORKING-STORAGE (РАБОЧЕЙ-ПАМЯТИ)
WRITE (ПИСАТЬ)
ZERO (НУЛЬ)
ZEROES (НУЛИ)
ZEROS (НУЛИ)

Глава 2. Описание данных

2.1. Оборудование

Важно сознавать, что вычислительная машина — это механизм. Знаменитый предшественник современных вычислительных машин, сконструированный в 1840 году, но так и не построенный, назывался аналитическим механизмом. Это название вполне соответствует действительности, так как, вопреки всем научно-фантастическим домыслам, вычислительная машина — это не гигантский мозг, а инструмент, предназначенный для обработки информации. Когда-нибудь действительно могут появиться интеллектуальные машины, так как ЭВМ — это особый вид машины, и уже сейчас имеются достижения в использовании ЭВМ для распознавания образов, игр, адаптивного программирования. Но при решении задач в области экономики вычислительные машины выполняют довольно простые операции. Их большим преимуществом перед человеком является только то, что они выполняют эти операции очень быстро и точно.

Структура вычислительных машин

Началом эры вычислительных машин можно считать 1947 год, когда Джон фон Нейман, математик Принстонского университета, высказал мысль, что память счетной машины может хранить не только данные, а также и команды, управляющие работой машины. Старые и простые настольные вычислительные машины имеют память, в которой может временно храниться два или три числа, но при их использовании последовательность операций задается извне и вручную (вводится число, нажимается клавиша сложения, вводится второе число, снова нажимается клавиша сложения, вводится еще одно число, нажимается клавиша результата). В настоящее время многие настольные счетные машины имеют программное управление, и с помощью одной внешней клавиши может быть задано от десяти до двадцати шагов. Такие настольные счетные машины наиболее близки к универсальным цифровым вычислительным машинам с хранимой сменной программой, которые вводят информацию, хранят и обрабатывают ее и выдают результаты пользователю.

Каждое действие, выполняемое такой вычислительной машиной, осуществляется под управлением числовых команд, хранящихся в ее памяти. Последовательность таких команд называется програм-

мой на языке машины. В далекие времена (около 1955 г.) программирование велось непосредственно на этом языке, при этом команды имели, например, такой вид:

111010110000101100000101010000110010

Эта 36-значная двоичная цифровая команда вызывает единственную машинную операцию, такую, как сложение одного числа с другим.

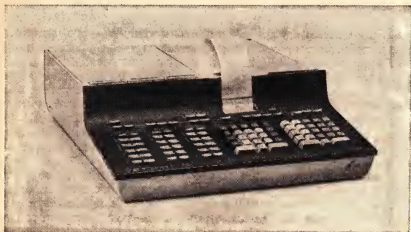


Рис. 2.1. Показанная на рисунке программируемая настольная вычислительная машина является, вероятно, наименьшей из большого семейства машин с хранимой программой, на которых извне можно задавать различные последовательности команд. Это машина фирмы Hewlett-Packard серии 9800.

И в настоящее время команды вычислительной машины имеют именно такой вид. В результате компиляции исходной КОБОЛ-программы строится рабочая программа, состоящая из таких команд. К счастью, эти команды могут вводиться в машину в закодированном виде, так что двоичные цифры могут быть представлены более коротко следующим образом:

24AC0012E

Каждая машинная команда состоит из двух основных частей: кода операции и адреса данных. Код операции определяет, какое действие машина должна произвести (например, помещение числа в регистр, который является сумматором). Эта операция подобна занесению числа в шкалу в механической клавишной машине. Адрес данных определяет, над каким числом будет производиться операция, так как он указывает, какое число, хранящееся в памяти, будет помещено в регистр. В связи с тем, что каждая команда управляет вы-

полнением только одного шага, программа на языке машины может состоять из тысяч команд. Часть программы на языке машины для определения наибольшего из трех чисел может потребовать четырнадцать шагов.

Как уже было сказано, вычислительная машина производит большое количество простых операций под управлением програм-



Рис. 2.2. Большая вычислительная система. На рисунке показаны некоторые из устройств. Соединительные кабели и каналы проходят под фальшполом. Фотография любезно предоставлена фирмой Burroughs.

мы, хранящейся в ее памяти, и идея хранения программ наряду с данными была выдвинута около 1947 года. Все это подразумевает, что в машине имеется определенное место для хранения этой информации. Большая часть физического объема вычислительной системы служит для хранения данных и передачи данных из одного устройства памяти в другое. В какой-то мере это свойство вычислительных машин «запоминать» информацию приводит к антропоморфическим сравнениям, так как человек имеет удивительную память. Именно память дает нам возможность учиться на опыте, проводить аналогии в новых ситуациях, короче говоря, быть мыслящими. Человек ско-

рее всего хранит все данные в своем мозгу в похожей форме, хотя точно не известно, каким образом. С другой стороны, вычислительная машина хранит информацию во многих местах и многими различными способами. Эти различные места взаимосвязаны каналами, или коммуникационными путями.

Кроме памяти, в машинах имеются арифметическо-логические блоки, которые производят простые арифметические действия (сложение, вычитание, умножение, деление) и простые сравнения (Больше ли одно число другого? Равно ли число нулю?). Во время второй мировой войны была необходимость автоматически и с большой скоростью производить арифметические расчеты, что дало толчок развитию вычислительной техники. В настоящее время такие потребности удовлетворяются машинами различных мощностей. Большие вычислительные машины выполняют эти операции за микросекунды, так что вполне возможно производить миллионы операций сложения в секунду, и делать это можно каждую секунду без ошибок. Имеется один шанс из десяти миллиардов, что машина сделает ошибку при сложении двух чисел. Это не значит, что результаты, выдаваемые вычислительной машиной всегда правильны, но источник ошибок обычно не в сбое самой машины, а в ошибках, допущенных человеком.

Команды, хранящиеся в памяти, выбираются для исполнения в основной блок управления. В свою очередь основной блок управления может направить часть этих команд для выполнения в блоках подуправления.

Запоминающие устройства

Программиста, пишущего на языке КОБОЛ, интересуют прежде всего те устройства, которые имеют дело с вводом данных в вычислительную систему и с хранением информации. Имеется два способа прямого ввода данных в систему непосредственно человеком: с помощью телетайпа или клавишного пульта и с помощью видеодисплея. В обоих случаях данные вводятся нажатием клавиш телетайпного типа, а результаты выдаются непосредственно в визуальной форме или на бумажную ленту, или на телеэкран. В настоящее время данные КОБОЛ-программы вводятся таким образом только на немногих установках. Все остальные методы ввода и вывода данных в вычислительной системе считаются непрямыми: между машиной и пользователем нет прямой связи. Простыми методами не прямой передачи данных являются способы ввода с помощью перфокарт и вывода с помощью печатающего устройства. Устройство чтения перфокарт читает файл карт (файл — это набор записей; каждая карта является записью), который был первоначально подготовлен человеком на перфорирующем устройстве. Печатающее устройство порождает файл (каждая запись является печатной стро-

кой; обычно такие устройства называются алфавитно-цифровыми печатающими устройствами) или неавтономно, в этом случае печатающее устройство непосредственно связано с машиной, или автономно, когда печатающее устройство не связано непосредственно с вычислительной машиной. В обоих случаях, при автономных и неав-



Рис. 2.3. На рисунке крупным планом показаны клавишный пульт и видеодисплей устройства непосредственного ввода информации. На заднем плане расположено устройство чтения перфокарт. Фотография любезно предоставлена фирмой Burroughs.

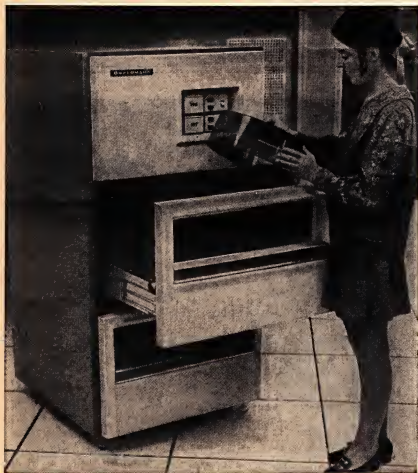


Рис. 2.4. Внешний вид дискового устройства. Пакеты дисков являются съемными. Время доступа к записи равно примерно десяти миллисекундам. Фотография любезно предоставлена фирмой Burroughs.

тоиомных печатающих устройствах, обычно используется промежуточная память. Автоиомное печатающее устройство обычно лучше, так как из помещения, где находится вычислительная машина, будет удален шум от печатающего устройства, пыль от бумаги и бумажные крошки.

К числу других методов иепрямой передачи данных отиосится передача данных с помощью магнитной ленты или магнитного диска. В обоих случаях данные записываются на магнитуую поверхность

подобно тому, как мы записываем песни на своем бобинном или кассетном магнитофоне. Фактически в машинах используются катушки размером с кассету. Диски напоминают по форме грампластинки, но вместо непрерывной спиральной дорожки, прорезанной на поверхности, запись на диске производится на замкнутых круговых и

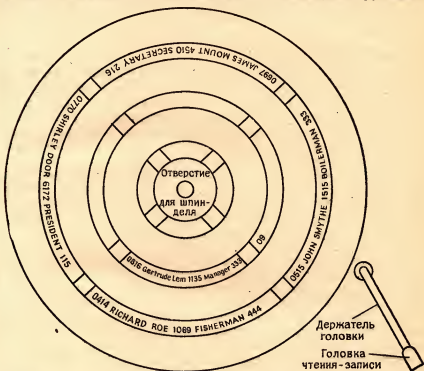


Рис. 2.5. Схематический вид расположения записей на диске. Показаны три concentric дорожки с четырьмя записями на каждой. На реальном диске могло бы быть, например, 200 дорожек и 100 записей на каждой дорожке. Хотя длина дорожек уменьшается с уменьшением радиуса, на каждой из них (и соответственно в каждой записи) располагается одинаковое количество литер за счет увеличения плотности записи. Держатель головки чтения записи движется, устанавливая головку на нужную дорожку.

concentric дорожках. Размер дисковой пластинки больше, чем размер обычной грампластинки.

Во всех случаях данные хранятся в том или ином закодированном виде. На перфокартах буквенно-цифровые литеры представлены в виде пробитых отверстий. Наличие или отсутствие отверстий в каждой из восьмидесяти колонок используется для обозначения букв, цифр или специальных литер. Колонка представляет собой

единственную позицию литеры. Каждая колонка имеет двенадцать вертикальных позиций, и именно комбинация отверстий в этих позициях представляет литеру. Тот же принцип используется в кодировании литер данных на магнитной поверхности лент. Это означает, что несколько дорожек проходят вдоль ленты и в каждой позиции литеры специальная комбинация магнитных импульсов кодирует буквенно-цифровую литеру.

Файлы КОБОЛа

В КОБОЛе термин *файл* относится к набору логических записей. Особо следует подчеркнуть, что речь идет именно о *логических* записях, так как в памяти машины имеются еще и *физические* записи. В некоторых случаях, как, например, на перфокартах, логическая запись совпадает с физической записью. В запоминающих устройствах с магнитной поверхностью, представляющей собой ленту или диск, одна физическая запись может содержать несколько логических записей. Файл тогда является как бы набором из нескольких карт, где каждая карта понимается как логическая запись. Аналогично данные в каждой логической записи рассматриваются как строка литер, напечатанных на карте одна за другой, как, например:

RICHARDROE777-56-9999FISHERMAN42

Хотя всякий файл есть не что иное, как набор записей, обычно не рекомендуется или бывает даже невозможно сгруппировать все обрабатываемые в вычислительном центре записи в один гигантский файл. Лучше иметь несколько файлов, каждый из которых содержит свой собственный набор записей, имеющих какую-то общую характеристику или общее назначение. Например, может быть составлен файл инвентаризации, содержащий записи, в которых указаны наименования товаров, артикулы, имеющиеся запасы, моменты возобновления заказов на запасы товаров. Может быть составлен отдельный файл для ежедневных сделок, содержащий запись для каждого поступления товара или каждого изъятия товара за данный день. Другой файл может содержать записи имен и адресов поставщиков, у которых можно заказать товары. Файлы могут быть разделены и различимы не только по «похожести» каждой записи внутри различных файлов, но также и по времени создания и обработки каждого файла. При решении экономических задач на машине весьма существенным является вопрос о структуре файла. Не просто также ответить на вопрос, какие и сколько записей включать в каждый данный файл.

Универсальные вычислительные машины действуют как машины, хранящие программу, и именно в память машины будет помещаться и там выполняться рабочая программа. Программа будет обрабатывать файлы путем считывания их записей в память машины.

Считывание не разрушает файлов, так что ленты и диски могут быть использованы многократно. Это аналогично повторному проигрыванию ленты на магнитофоне. Программа может также создавать файлы, собирая данные в записи и занося их в определенный файл. Процесс занесения записи разрушителен, и то, что было в файле раньше, уничтожается, так же, как запись новой песни на магнитофоне стирает старую. Большую опасность представляет занесение записи не в тот файл. Принимается много мер предосторожности, чтобы избежать разрушения важной информации.

В процессе считывания файлов и занесения записей формируется понятие внешнего и внутреннего: так, все файлы считаются внешними, даже если данные могут быть расположены на дисках, которые являются постоянной частью оборудования машины. Информация из этих внешних файлов недоступна для программы до тех пор, пока к записи не будет получен доступ, и она не будет считана во внутреннюю память. Большинство файлов имеет последовательный доступ, так что доступ ко второй записи может быть получен не раньше, чем прочитана первая запись, к третьей не раньше, чем прочитана вторая, и т. д. 5000-я запись в файле с последовательным доступом не может быть прочитана раньше, чем будут прочитаны предыдущие 4999. Этот аспект обработки файлов влияет на разработку программных процедур. Имеется и другая организация доступа, называемая произвольной выборкой данных, но она будет описана в гл. 10.

Когда логическая запись передается из внешнего файла во внутреннюю память, она попадает в область внутренней памяти, называемую файловой областью. При компиляции раздела DATA DIVISION (РАЗДЕЛ ДАННЫХ) часть этой файловой области выделяется для хранения записей каждого файла, описанного в статье FD. Эта область записи выделена именно этому файлу, и никакой другой файл не может ее использовать. Это означает, что программа не может считать запись из одного файла и записать сразу же эту запись в другой файл, так как для второго файла будет выделено в файловой области отдельное место. При переписи записей из первого файла во второй прежде, чем запись сможет быть записана во второй файл, необходимо переместить ее из области записи первого файла в область записи второго файла. На рис. 2.6 схематически показана эта взаимосвязь. Позже, в гл. 6, будет описано специальное исключение из этого соглашения. Кроме того, когда файл организован по принципу последовательного доступа, каждая запись во время ее считывания в файловую область проходит позицию чтения в устройстве, на котором расположен файл (например, в устройстве чтения перфокарт или лентопротяжном устройстве). При этом следующая запись устанавливается в позицию чтения. Таким образом, программист не может считать запись, обновить ее и затем записать в тот же самый файл, не разрушив следующей записи этого файла.

В КОБОЛе не существует команды возвращения файла в предыдущее положение. Обычно обработка файлов в КОБОЛе состоит из считывания одного или более файлов, перемещения и модификации записей во внутренней памяти и переписи их в другой файл.

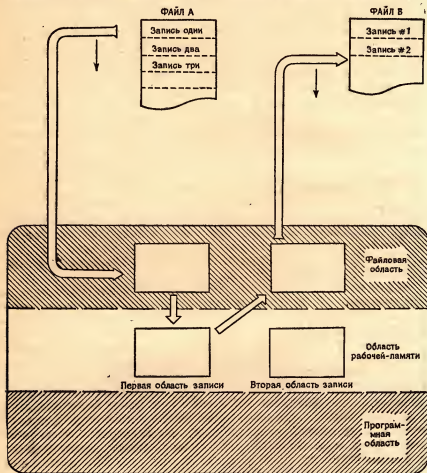


Рис. 2.6. Понятия файла и записи.

Кроме файловой области, внутренняя память имеет другую область, называемую областью рабочей-памяти. Промежуточные записи или константы, в которых нуждается программа, хранятся в этой области внутренней памяти. Область рабочей-памяти также доступна для программы, как и файловая область, но ни считывание, ни запись не могут производиться непосредственно в область

рабочей-памяти или из нее. Некоторые программы будут считывать запись в файловую область, обрабатывать или обновлять ее непосредственно в ней, а затем перемещать запись в другую область записи файловой области, прежде чем записать результат в другой файл. Другие программы считают запись, затем переместят ее в область рабочей-памяти, обработают ее там и переместят запись для вывода в область записи в файловой области. Еще одна возможность заключается в том, чтобы считать запись в файловую область, переместить ее на другое место в этой области и обработать ее там, прежде чем вывести обновленную запись. Выбор последовательности, которой следует придерживаться, зависит от конкретной задачи.

Обычно начинающим не нужно знать, каким образом происходит физическое хранение данных в вычислительной системе, но некоторые общие объяснения безусловно помогут мысленно представить себе оборудование, на котором работает программа. Внутренняя память вычислительной машины относительно невелика, она может хранить около 5000 данных вместе с рабочей программой, но скорость ее работы очень высока и исчисляется микросекундами. Внутренняя память является памятью с *единым временем доступа*; это означает, что независимо от того, где хранятся записи данных, требуется одинаковое время для извлечения любой порции памяти. Никакая информация не может быть обработана, пока она не помещена во внутреннюю память, и очень желательно иметь там как можно больше данных с тем, чтобы использовать преимущества чрезвычайно быстрого доступа. Внутренняя обработка данных, такая, как арифметические операции, сравнения, пересылки данных, также осуществляется с теми же высокими скоростями.

Прежде чем записи данных смогут быть переданы из внешней памяти во внутреннюю, к ним надо получить доступ; это означает, что физическая область, в которой они хранятся, должна быть установлена в позицию считывания. Для того чтобы проверить готовность физического устройства и переслать информацию по каналу во внутреннюю память, требуется некоторое время. Скорость передачи достаточно велика (от 50 000 до 200 000 бит в секунду), но она мала по сравнению с микросекундными скоростями выполнения внутренних операций. Если внутренняя память характеризуется как небольшая по объему, но быстрая, то внешняя память характеризуется как большая по объему и медленная. В сумме времени доступа и передачи могут составить от 1 до 100 миллисекунд: это может быть в несколько тысяч раз медленнее внутренних скоростей. С другой стороны, внешняя память — это массовая память: в ней могут храниться миллионы данных. Необходимо также рассмотреть проблему доступности канала. Канал — это связующая магистраль передачи данных. Короче говоря, одновременно только одно устройство может передавать информацию по данному каналу. Все остальные устройства хранения файлов должны ждать, пока канал не освободится.

дится. Проблемы распределения каналов, планирования ввода и выделения устройств имеют первостепенное значение при системном программировании.

Устройства массовой памяти могут быть представлены двумя типами устройств: лентопротяжными механизмами протяжки магнитных лент и дисковыми устройствами, обычно называемыми просто

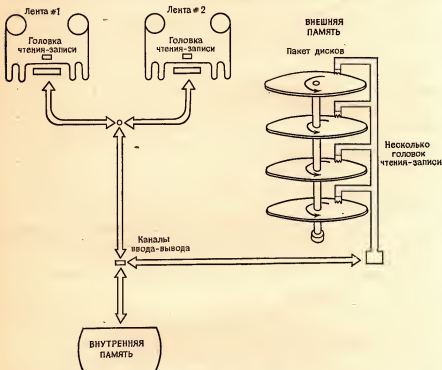


Рис. 2.7. Типы устройств.

ТАРЕ (ЛЕНТА) и DISK (ДИСК). На ленте данные представлены в виде магнитных импульсов, записанных вдоль нескольких дорожек, сгруппированных в физические записи, разделенные промежутками между записями в четверть дюйма. На пакете дисков данные также представлены магнитными импульсами, но записанными вдоль узкой круговой дорожки на диске, напоминающем по виду большую граммофонную пластинку. Каждая физическая запись ограничена одной окружностью, так что для хранения очень большой логической записи может потребоваться несколько физических записей. Для дисковой памяти существует какое-то время доступа, в то время как для памяти на магнитной ленте такое время отсут-

ствуется, так как следующая запись готова для считывания сразу же после того, как прочитана предыдущая. Для простого программирования на базовом КОБОЛе не имеет особого значения, какое устройство будет использовано для хранения файла; при системном программировании устройство и метод обработки должны быть тщательно согласованы.

На рис. 2.7 схематически показаны два типа устройств и возможные ограничения на каналы ввода-вывода. Реальная установка может иметь восемь дисковых устройств, десять лентопротяжных устройств и шесть каналов, но распределять устройства и каналы придется всегда.

Упражнения

1. Напишите определения:
 - а) рабочей программы, хранящейся в памяти;
 - б) адреса данных;
 - в) файловой области;
 - г) занесения в файл;
 - д) последовательного доступа;
 - е) области рабочей-памяти;
 - ж) времени передачи;
 - з) времени доступа;
 - и) промежутка между записями;
 - е) каналов данных.
2. Каковы характеристики:
 - а) внутренней памяти в противоположность внешней памяти;
 - б) прямого ввода данных в противоположность непрямоу вводу.
3. Имеются три предложения из параграфа:

PARA-DATA-MOVE.

READ INPUT-FILE-6 RECORD

AT END CLOSE INPUT-FILE-6 STOP RUN.

MOVE INPUT-RECORD TO PRINT-RECORD.

WRITE PRINT-RECORD.

Нарисуйте схему файловой области и покажите перемещения введенной информации.

2.2. Требования к вычислительной машине

Создатели языка КОБОЛ надеялись, что между различными вычислительными машинами может быть достигнута большая степень совместимости, т. е. что программа, написанная на исходном языке КОБОЛ, может компилироваться или транслироваться на язык лю-

бой рабочей машины. Однако признано, что такая цель никогда не может быть достигнута в полной мере из-за различий в вычислительных машинах и из-за различий во времени и способах создания различных компиляторов. Для обособления и упрощения внесения изменений, необходимых для достижения совместимости, были предназначены разделы ENVIRONMENT DIVISION (РАЗДЕЛ ОБОРУДОВАНИЯ) и DATA DIVISION (РАЗДЕЛ ДАННЫХ). В разделе ENVIRONMENT DIVISION изменения будут касаться имени устройства в статье

SELECT имя-файла ASSIGN TO имя-устройства.

Выбор этого имени будет определяться реализацией языка или конкретной установкой. В разделе DATA DIVISION имеются определенные статьи, которые специфицируются как необязательные в стандартном КОБОЛе, но могут потребоваться на какой-то конкретной установке. Программист, пишущий на КОБОЛе, должен выяснить, какие требования предъявляются на конкретной установке, и следовать им; это не будет сводиться просто к добавлению всех необязательных вариантов, так как в некоторых случаях они могут быть неверны.

Фраза LABEL (МЕТКИ) служит для описания того факта, имеет ли файл дополнительно к записям, описанным программистом, одну или более специальных записей меток. Эти записи содержат такую информацию как имя файла, дату создания, как долго следует сохранять файл и кому принадлежит файл. Эти стандартные записи меток готовятся операционной системой вычислительной машины и являются важным вспомогательным средством в идентификации и защите файлов пользователя. Для любого файла, созданного программой, всегда должна быть написана фраза:

LABEL RECORD IS STANDARD.

Таким образом будет установлено, что все файлы имеют стандартные метки и будут защищаться секцией управления вводом-выводом операционной системы. Только в двух ситуациях следует использовать вариант OMITTED (ОПУЩЕНЫ): если файлы подготовлены какой-либо программой, в которой не задан вариант STANDARD, и, таким образом, не имеют никаких стандартных меток или когда файл размещен на устройстве считывания карт или телетайпе, так как карты и вводимые с телетайпа строки не будут в большинстве случаев иметь записей меток. Однако с появлением в машинах параллельной обработки эти файлы все в большей степени требуют какой-либо идентифицирующей записи. Таким образом, программистам, пишущим на языке КОБОЛ, рекомендуется использовать вариант STANDARD всегда, кроме случаев, когда возникают специфические для данной установки трудности.

Дополнительная фраза, которая еще не упоминалась, но может потребоваться в статье FD (ОФ) при компиляции в некоторых вычислительных центрах — это фраза

VALUE OF специальное-слово IS литерал.

Эта фраза не является фразой VALUE, используемой для задания начальных значений данных в статье-описания-данного в рабочей-памяти. Различие состоит в ключевом слове OF. Назначение фразы VALUE OF состоит в том, чтобы указать блоку управления вводом-выводом операционной системы на необходимость проверки того, совпадает ли значение специального слова в метке файла со значением литерала. Это специальное слово должно быть фиксированным именем, определяемым авторами компилятора, и не может быть выбрано программистом. Проверьте, каково это специальное слово на данной вычислительной установке или для данного компилятора. Во многих случаях этим словом будет IDENTIFICATION или ID. Литерал является именем и может создаваться программистом. Он используется для идентификации определенного файла и хранится в стандартных метках этого файла. Примером такой фразы является:

VALUE OF ID IS "AM305"

Значение этого литерала часто требуется в качестве еще одного идентификатора файла в дополнение к имени-устройства и имени-файла КОБОЛа. Это может показаться избыточным, но каждое имя имеет свое назначение. Связь между различными именами показана в следующей главе при описании глаголов OPEN и READ. Значением литерала во фразе VALUE OF должно быть правильное слово КОБОЛа, но для обеспечения более широкой совместимости следует придерживаться правила, что слово должно состоять из семи или менее литер, не должно содержать дефисов и должно начинаться с буквенной литеры (исключая пробелы). Когда программа создает выходной файл, фраза VALUE OF определяет значение специального-слова при формировании стандартной записи меток. Фраза VALUE OF является необязательной в КОБОЛе, но она может быть необходимой в определенных реализациях языка.

В дополнение к этим возможностям, которые разрешаются в спецификации языка КОБОЛ, могут быть компиляторы, которые не полностью согласуются со стандартными требованиями. В частности, в статье FD может потребоваться другая или специальная фраза. За исключением этих различий, которые могут показаться начинающему столь раздражающими, КОБОЛ является языком стандартизированным и широко распространен в различных странах. После приобретения некоторого опыта работы с языком эти различия покажутся столь же незначительными, как различия в диалектах естественного языка в различных частях одной страны.

Программа не может считаться законченной до тех пор, пока она не будет оттранслирована на язык машины, загружена в память машины и выполнена. Язык КОБОЛ — это полезный язык, на котором пишется программа, но пользователь должен владеть

```
//JOBNAME JOB (DRXL,4223),'USER-NAME',MSGLEVEL=1
// EXEC COBUCLG,PARM=QUOTE
//COB.SYSIN DD *
  IDENTIFICATION DIVISION.
  PROGRAM-ID. TEST.
  ENVIRONMENT DIVISION.
  CONFIGURATION SECTION.
  SOURCE-COMPUTER. IBM-370.
  OBJECT-COMPUTER. IBM-370.
  INPUT-OUTPUT SECTION.
  FILE-CONTROL.
    SELECT CARD-INPUT-FILE ASSIGN TO UT-S-SYSIN.
    SELECT CARD-IMAGE-FILE ASSIGN TO UT-S-AM669.
  DATA DIVISION.
  FILE SECTION.
  FD CARD-INPUT-FILE
    LABEL RECORD IS OMITTED.
  01 CARD-INPUT-RECORD PICTURE IS X(80).
  FD CARD-IMAGE-FILE
    LABEL RECORD IS STANDARD.
  01 CARD-AGE-RECORD PICTURE IS X(80).
  PROCEDURE DIVISION.
  BEGIN-X.
    OPEN INPUT CARD-INPUT-FILE.
    OPEN OUTPUT CARD-IMAGE-FILE.
  LOOP-X.
    READ CARD-INPUT-FILE RECORD
      AT END CLOSE CARD-INPUT-FILE CARD-IMAGE-FILE STOP RUN.
    WRITE CARD-IMAGE-RECORD FROM CARD-INPUT-RECORD.
    GO TO LOOP-X.
//GO.AM669 DD DSN=AM669,UNIT=SYSSQ,SPACE=(TRK,100),DISP=(NEW,KEEP)
//GO.SYSIN DD *
  THIS IS THE FIRST DATA CARD.
  THIS IS THE SECOND DATA CARD.
  THIS IS THE THIRD DATA CARD.
//
```

Рис. 2.8. Управляющие карты, необходимые для компиляции, загрузки рабочей программы и задания последовательности исполнения для вычислительной системы. Приводимый пример предназначен для установки, использующей операционную систему IBM OS/360-370.

еще одним языком, хотя бы его частью. Это язык команд операционной системы, управляющей работой вычислительной машины. Современными высокоскоростными вычислительными машинами не управляют вручную, хотя и при их использовании ряд операций выполняет человек (устанавливает катушки с магнитной лентой или дисковые пакеты). Но все же машины управляются с помощью

программ, хранящихся в памяти. Управляющие программы, с помощью которых машина эксплуатируется, называются операционной системой и имеют свой собственный язык команд. К счастью, начинающему программисту приходится использовать лишь небольшое число команд операционной системы, необходимых для запуска процесса компиляции, загрузки рабочей программы и идентификации файлов и устройств, на которых расположены файлы. Если исходная КОБОЛ-программа пробита на перфокартах, то ей должны предшествовать и за ней должны следовать управляющие карты. Эти карты неизменно будут содержать несколько специальных литер, отперфорированных в первых позициях для того, чтобы выделить их. Например, последовательность записей может быть такой:

```
? COMPILE COBOL; DATA DECK. MAGINNIS
      (Программная колода на языке КОБОЛ)
? DATA = AM305
      (Файл записей данных ввода)
? END
```

Записи с первой литерой ? являются управляющими записями и не являются частью языка КОБОЛ. Другой пример для другой вычислительной машины и другой операционной системы может выглядеть следующим образом:

```
// EXEC COB3CLG
// COB.SYSIN DD *
      (Программная колода на языке КОБОЛ)
// GO.SYSIN DD *
      (Файл записей данных ввода)
/*
```

Записи с первыми литерами // и /* являются управляющими записями. Третий пример приведен на рис. 2.8, на котором показаны управляющие карты и программа на языке КОБОЛ для чтения записей и внесения их в печатный файл. В этом примере показано расположение записей данных, но сами эти записи не приведены.

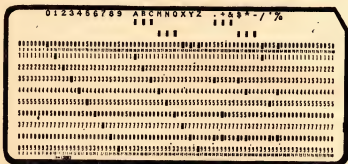
2.3. Представление значений

Вся информация, хранимая в вычислительной системе, независимо от того, внешняя она или внутренняя, будь это программа или данные, кодируется как серия *битов*, т. е. цифр 0 или 1. Бит (сокращение слов «binary digit» (двоичная цифра)) может принимать только два значения 0 и 1. Такое двоичное кодирование не означает, что машина производит вычисления только в двоичной арифметике

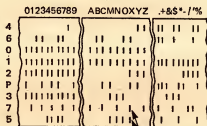
или что могут храниться только цифры. Это означает, что если бы память можно было увидеть реально (как при прогоне магнитной ленты через водяную взвесь железных частиц), то все, что можно было бы наблюдать, — это последовательность вида

1010001011010010010

или какую-либо ее физическую аналогию. Таким образом, запись — это такой поток битов. Запись имеет фиксированное начало и опре-



Девять дорожек



Вертикальные черточки представляют двоичную цифру 1

Рис. 2.9. Пример кодирования литер на перфокарте и магнитной ленте. На карте имеется двенадцать дорожек и восемьдесят позиций литер; число дорожек на ленте равно девяти, а число позиций литер не ограничено.

деленный конец, но между ними невозможно заметить путем наблюдений никаких границ. Кроме того, кодировка одних и тех же данных может меняться при перемещении записи из внешней памяти во внутреннюю.

Информация во внутренней памяти всегда представляется в виде литер. Определенные группы битов используются для обозначения

цифр или специальных литер (таких, как %, +, \$ и т. д.). Таким образом, приведенный выше поток битов мог бы кодировать слово:

CAT

Даже пробел или пропуск имеет свой собственный код. В зашифрованном потоке не может быть пропущенных битов. Но программист, пишущий на КОБОЛе, может мысленно представлять внешнюю запись в виде непрерывного набора литер, как в этих двух примерах:

0010THIS IS A RECORD4100321XX592365

0020SO IS THIS GROUP7506135YY600596

Каждая из записей в этих примерах содержит по тридцати пяти литер, включая пробелы, и может потребовать для кодирования 280 битов. Информация во внутренней памяти кодируется гораздо более сложным образом, различным для разных вычислительных машин. Программиста, пишущего на КОБОЛе, не должна заботить эта сложность, и он может мысленно представлять числа в виде десятичных цифр, а буквенные литеры в виде заглавных букв. Использование некоторых фраз КОБОЛа, особенно фразы USAGE IS (ДЛЯ), описанной в гл. 9, позволит опытному программисту указать машине на выбор наилучшего кода для хранения данных во внутренней памяти.

Числовые значения данных, представленных в записях внешних файлов, могут иметь знак, т. е. они могут быть положительными или отрицательными. Числовые значения не обязательно должны иметь знак, во многих случаях знак опускается и для вычислений используется только абсолютное значение. Однако, если данные имеют знак, символ плюс или минус может быть помещен в отдельную позицию литеры в записи, обычно непосредственно перед числом, так что придется оперировать с числами вида +32 или -948. Иногда, но значительно реже, знак помещается непосредственно после числа, например: 32+ или 948—. Имеется также еще одна возможность, при которой знак помещается в самую правую позицию литеры вместе с самой правой цифрой:

32⁺ 948⁻

Комбинация цифры и знака, закодированных в одной и той же позиции литеры, порождает комбинацию битов, которая не является ни цифрой, ни знаком, а представляет собой некий новый код. Иногда этому новому коду не соответствует никакая печатаемая литера, а иногда — это буква. Поэтому на некоторых вычислительных машинах реальные строки литер для этих двух примеров могли бы быть такими:

3В 94Q

Такое «совмещение» знака и самой правой цифры быстро выходит из употребления на многих вычислительных установках, но тем не менее все еще существует, особенно там, где сохраняется ориентация на использование карт. Если знак есть, то независимо от того, где он расположен, он называется знаком числа. Язык КОБОЛ предназначен для обработки любой из этих ситуаций, с отдельным знаком или с совмещенным знаком; при этом предполагается, что число снабжено знаком правильно. Во многих случаях при программировании экономических задач обычно стараются не использовать чисел со знаками во внешнем представлении.

На перфокарте результат совмещения цифры и знака легко различим, так как цифре два соответствует одно отверстие, пробитое во второй строке, в то время как кодом знака плюс является одно отверстие, пробитое в двенадцатой строке. Совмещение цифры два и знака плюс в одной и той же позиции порождает такую же кодовую комбинацию, как для буквы В. В терминах магнитных импульсов этот результат не так легко описать, однако он оказывается тем же.

В логической записи, хранимой во внешней или внутренней памяти, одна или более позиций литер могут быть идентифицированы программистом как *элементарное данное*. Это делается с использованием статьи-описания-данного в DATA DIVISION и будет нами сейчас рассмотрено. Начало и конец элементарного данного произвольны. Например, в двух записях из 35 литер, приведенных выше, первые четыре позиции могут быть идентифицированы как элементарное данное CARD-NUMBER и будут иметь значение 0010 в первом случае и 0020 во втором. Другим элементарным данным могли бы быть позиции от 5 до 20, для записи 0010 это

THIS IS A RECORD

Понятие элементарного данного позволяет программисту присвоить имя последовательности литер в записи и обрабатывать ее как единое целое. Одна и та же запись может одновременно подразделяться на различные элементарные данные, так что одна и та же запись может обрабатываться различными путями, например:

0010/THIS IS A RECORD/4100/321/XX59/2365

или

001/0/THIS IS A RECORD/41003/21XX/592365

или

0010/THIS IS A RECORD/4100321XX592365

Знаки деления "/" в записи не присутствуют; они приведены для наглядности. Разбиение позиций на элементарные данные производится с помощью отсчета позиций литер слева направо.

Элементарное данное может иметь два типа значений: числовое и нечисловое. Числовое значение состоит из строки десятичных цифр, с которой может быть связан знак, иногда совмещаемый с цифрой в последней позиции. Числовое значение не может содержать явную десятичную точку. Это не означает, что значения ограничиваются целыми значениями, а означает лишь то, что фактически могут храниться только цифры. Использование дробей допускается с помощью подразумеваемой десятичной точки, которую программа запоминает, когда имеет дело с арифметикой.

Примерами числовых данных являются

932 3В 94Q 5689654102

Так как второй и третий примеры оканчиваются буквенными литерами, они, вероятно, являются числами со знаками. Другие два примера являются числами без знаков и будут рассматриваться в любых арифметических операциях как положительные.

В этих четырех примерах числовых данных нет десятичных точек, и просто, глядя на данное само по себе, невозможно определить, представляет ли 932 число 932 или 93.2, или 9.32 или даже 932.000. Подразумеваемая десятичная точка будет указана во фразе описания, которая должна быть связана с каждым элементарным данным. Фраза PICTURE (ШАБЛОН), задающая такое описание, будет рассмотрена ниже.

Другой тип значений, которые может принимать элементарное данное,— это нечисловые значения. Нечисловое данное состоит из строки любых литер, допустимых на данной вычислительной машине, т. е. из буквенно-цифрового набора литер. Нечисловые данные могут содержать в качестве составных частей цифровые номера, но эти цифры нечисловые и не могут быть использованы в арифметических операциях. Примерами нечисловых данных являются

THIS IS A RECORD

\$ 569.32

508-18-5677

ANSI X3.23-1973

10091975

Хотя нечисловые данные не могут быть использованы в арифметических вычислениях (ADD, MULTIPLY и т. д.), они могут сравниваться друг с другом. Для упорядочения нечисловых данных используется буквенная последовательность, так что слово CAT считается «больше», чем слово BAT.

Длина и тип элементарного данного определяется в программе фразой КОБОЛА, начинающейся словом PICTURE. Фраза PICTURE указывает также позицию подразумеваемой десятичной точки в числовых данных и управляет процессом редактирования, исполь-

зуемым для того, чтобы сделать выходные данные более удобочитаемыми. Общий формат для определения фразы PICTURE таков:

PICTURE IS строка-литер

Символы, используемые в строке-литер, описывают данное и управляют операциями программы с участием этого данного. Фраза PICTURE должна быть включена в статью-описания-данного для каждого элементарного данного, описанного в разделе DATA DIVISION. Эти статьи должны существовать для каждого данного, упоминаемого в КОБОЛ-программе. Примером фразы PICTURE является:

PICTURE IS XXXXXX

Шесть символов X означают, что данное нечисловое и состоит из шести литер. Если значение данного должно быть определено как числовое и целое, тогда следует использовать фразу:

PICTURE IS 999999

где символы 9 определяют, что в шести позициях литер встречаются цифры. Символы 9 не задают значение самого данного, а только его формат; значения, которые может принимать это данное, могли бы, в частности, быть такими: 000001, 123456 и даже 999999, но полученными не из самой фразы PICTURE.

Этот простой способ использования фразы PICTURE довольно очевиден. Существует более сложное применение этой фразы, связанное с оператором MOVE (ПОМЕСТИТЬ), которое будет описано в гл. 3. До сих пор мы познакомились с двумя типами данных, описываемых с помощью фразы PICTURE: нечисловые данные и числовые данные. Для описания нечислового данного строка-литер, следующая за словами PICTURE IS, может содержать только символ X. Число символов X в этой строке-литер определяет длину данного. Например, описание:

01 EXAMPLE-ITEM PICTURE IS XXX.

означает, что данное EXAMPLE-ITEM может принимать нечисловые значения типа:

ABC
\$10
.05
(*)

но должно содержать не больше и не меньше трех буквенно-цифровых литер.

Строка литер для числового данного может содержать символы 9, и количество символов 9 определяет длину данного. Однако кроме символов 9 в строке PICTURE для числовых данных допускается использование символов S (3), V (T) и P (M). Эти символы не учитываются при определении размера элементарного данного, а используются для обозначения знака и местоположения подразумеваемой десятичной точки.

Буква S используется для указания наличия знака в данном, и если она есть, то она должна быть самой левой литерой в строке-литер фразы PICTURE. Это означает, что буква S должна записываться как самая левая литера, например:

PICTURE IS S999.

Если символ S не используется, то данное будет храниться в памяти как абсолютное значение.

Символ V используется в строке-литер фразы PICTURE для указания местоположения подразумеваемой десятичной точки (запомните, что явная десятичная точка не может фактически присутствовать в числовых данных) и может появляться только один раз. Символ V может появляться в любой позиции строки-литер, но может быть опущен, если он расположен в самой правой позиции. Так, например, описание:

01 NUMERIC-ITEM PICTURE IS 999V.

имеет подразумеваемую десятичную точку в самой правой позиции, а данное NUMERIC-ITEM может содержать только целые трехзначные числа. Но такая спецификация полностью совпадает просто с

01 NUMERIC-ITEM PICTURE IS 999.

Примерами использования символа V являются:

01 EXAMPLE-ONE PICTURE IS 99V9.

01 EXAMPLE-TWO PICTURE IS V99999.

Символ P используется для представления подразумеваемых нулей, когда подразумеваемая десятичная точка V находится вне цифр данного. Например, данное

01 BIG-NUMBER PICTURE IS 999PPP.V.

будет содержать только три цифры, так как P и V (и S) реально не представляют позиций литер в данном. Если данное имеет значение 932, то оно будет означать 932000. Ниже следуют примеры числовых строк-литер:

Литеры, фактически хранящиеся в памяти машины	Строка-литер фразы PICTURE, описывающая данное	Значение, определенное данным
932	S999	+932.
932	S9V99	+9.32
932	S99V9	+93.2
932	S999V	+932.
932	VPP999	.00932
93K	S999	-932.
93K	S9V99	-9.32
93K	S99V9	-93.2
93K	S999V	-932.
93K	SVPP999	-.00932

Символ V может также быть опущен, когда используются символы P, так как они неявно задают местоположение подразумеваемой десятичной точки, как во фразе PICTURE IS PP99, которая показывает, что данное должно быть чем-то вроде .0012.

Максимальное число литер, допустимое в строке-литер фразы PICTURE (такой как S99V99), равно тридцати, но это не ограничивает описательные возможности, так как для фразы PICTURE существует правило повторения. Это правило гласит, что любая литера строки-литер шаблона, за которой следуют круглые скобки, рассматривается как повторяющаяся столько раз, сколько указано в скобках. Таким образом:

PICTURE IS X(6)

то же самое, что

PICTURE IS XXXXXX

а

PICTURE IS VP(2)9(3)

то же самое, что

PICTURE IS VPP999

Для данных также существуют ограничения размера: для числовых данных — это восемнадцать десятичных цифр, а для нечисловых — 120 литер. Эти размеры чаще всего вполне достаточны.

2.4. Групповые данные

Запись может состоять целиком из одного элементарного данно-го. В таком случае статья-описания-записи может выглядеть следующим образом:

01 CARD-IMAGE PICTURE IS X(80).

Это означает, что именем записи является CARD-IMAGE и что она является элементарным нечисловым данным длиной в 80 позиций. В качестве другого примера можно привести описание:

01 TODAYS-DATE PICTURE IS 9(6).

которое означает, что имя записи — это TODAYS-DATE, ее длина — 6 цифр; данное не имеет знака или десятичной точки и является, таким образом, положительным целым числом. Эти записи легко идентифицируются как элементарные данные, благодаря наличию фразы PICTURE, которая может и должна присутствовать только в описании элементарных данных. Но запись может состоять более, чем из одного элементарного данного. Набор смежных элементарных данных называется *групповым данным*. Запись с двумя и более элементарными данными является групповым данным, как, например:

01 SOME-INPUT-RECORD.

05 CARD-IDENTIFICATION	PICTURE IS 999.
05 TODAYS-DATE	PICTURE IS 9(6).
05 PERSONS-NAME	PICTURE IS X(25).

Это описание показывает, что в записи с именем SOME-INPUT-RECORD позиции с первой по третью отведены для данного CARD-IDENTIFICATION, позиции с четвертой по девятую — для данного TODAYS-DATE и позиции с десятой по тридцать четвертую — для нечислового элементарного данного PERSONS-NAME. Групповые данные могут быть в свою очередь объединены в группы из двух или более групповых данных.

При построении таких комбинаций смежных данных возникает необходимость придерживаться определенной организации. Так как записи являются самыми объемлющими объединениями данных, им всегда отводится уровень 01. Менее объемлющим элементам в составе записи отводятся более высокие номера уровней, хотя эти номера не должны обязательно отводиться подряд. На деле для более простого внесения исправлений в программу разрешаются пропуски при выборе номера уровня. Номера уровней являются двузначными числами от 01 до 49. Эти номера не должны обязательно иметь впереди нуль: 01 может быть записано просто как 1, а 05 может быть записано как 5 и т. д. В этой книге всегда будут использоваться двузначные номера уровней. И этого рекомендуется придерживаться. Имеются три специальных номера уровня: 66, 77 и 88, но они будут описаны немного позже. Для каждого используемого номера уровня пишутся отдельные статьи КОБОЛ-программы.

Группа включает все группы и элементарные данные следующие за ней до тех пор, пока не встретится номер уровня равный или меньший, чем номер уровня этой группы. Групповое данное

должно иметь имя. Статьи-описания-записи с номером 01 должны начинаться в поле А, при этом следующее за номером уровня имя-записи должно располагаться в поле В. По крайней мере один пробел должен отделять номер уровня от слова, следующего за ним.

Для того чтобы сделать более ясной иерархическую организацию программы, программистам, пишущим на КОБОЛе, приходится пользоваться отступами (как показано во всех примерах данной книги). Отступ не влияет на величину номера уровня и используется только для того, чтобы создать удобство для зрительного восприятия программы пользователем. Ниже приводится пример описания записи, показывающий несколько уровней подразделения:

01 EXAMPLE-RECORD-ENTRY.

05 CARD-IDENTIFICATION	PICTURE IS 999.
05 TODAYS-DATE.	
10 MONTH-X	PICTURE IS 99.
10 DAY-X	PICTURE IS 99.
10 YEAR-X	PICTURE IS 99.
05 PERSONS-NAME.	
10 LAST-NAME	PICTURE IS X(15).
10 REST-OF-NAME.	
15 PART-OF-NAME	PICTURE IS X(9).
15 MIDDLE-INITIAL	PICTURE IS X.

Размер записей, описанных в этой статье, составляет тридцать четыре позиции, так же как и в предыдущих примерах. Групповое данное EXAMPLE-RECORD-ENTRY подразделяется на элементарное данное CARD-IDENTIFICATION, за которым следует два групповых данных TODAYS-DATE и PERSONS-NAME. Первое подразделяется на три элементарных данных, в то время как PERSONS-NAME состоит из элементарного данного, за которым следует групповое данное. Все элементарные данные имеют фразу PICTURE. Каждая статья заканчивается точкой со следующим за ней пробелом. Имейте в виду, что пробелы перед фразой PICTURE делаются для того, чтобы статья-описания-записи была более удобочитаемой, и не влияют на ее смысл. Обратите внимание на использование имен данных с дефисами, что позволяет избежать употребления зарезервированного слова в качестве имени, а также придать как можно больше смысла идентифицирующему имени. Запись, удовлетворяющая этому описанию может выглядеть следующим образом:

015120673SMITHSONIAN ROBERT L

Каждой группой и поименованным элементарным данным можно оперировать как отдельным данным, так что данное MONTH-X или

TODAYS-DATE могут использоваться как две цифры или шесть цифр при внутренних перемещениях или обработке.

Каждое групповое данное должно иметь уникальное имя (дубликатов имен быть не может). Однако предоставляется метод для удовлетворения требования наличия имен элементарных данных,

CARD IDENTIFICATION	TODAYS-DATE			PERSONS-NAME		
	MONTH	DAY	YEAR	LAST-NAME	REST-OF-NAME	
	-X	-X	-X		PART-OF-NAME	MIDDLE-INITIAL
698122375SCHMIDLAP ROBERT M						

Рис. 2.10. Схематическое расположение и пример записи, состоящей из 34-х литер и описанной с помощью статьи-описания-записи с именем EXAMPLE-RECORD-ENTRY.

которые не будут обрабатываться, но должны присутствовать, чтобы отделять другие данные. Если бы одно данное длины 3 нужно было расположить в колонках 1—3, а второе данное нужно было расположить в колонках 78—80 образа карты, то для их правильного расположения необходимо было бы описать данное длиной в 74 позиции, находящееся между ними. В таком случае в качестве имени фиктивного данного может использоваться слово FILLER (ЗАПОЛНИТЕЛЬ):

01 CARD-IMAGE.

05 FIRST-ITEM	PICTURE IS 999.
05 FILLER	PICTURE IS X(74).
05 SECOND-ITEM	PICTURE IS 999.

К данному FILLER никогда нельзя обратиться и оно должно иметь фразу PICTURE с символами X, чтобы избежать отказов при вводе нецифровых литер. Возможность FILLER предназначена для уменьшения количества имен в программе.

Упражнения

1. Предположим, что каждая из десяти цифр для хранения в памяти машины закодирована четырехбитовым двоичным кодом:

	0—0000	
1—0001	4—0100	7—0111
2—0010	5—0101	8—1000
3—0011	6—0110	9—1001

Запишите строку битов, которая будет храниться в машине кодом стандартной статьи данных:

05121974

2. Классифицируйте следующие стандартные данные по их типу (числовые или нечисловые):
- а) 1976
 - б) 3.146
 - в) \$14.32
 - г) MAXIMUM
 - д) 6,432
 - е) 98
 - ж) —93
 - з) 39A
3. Вставьте пропущенные (подчеркнутые) значения в следующей таблице:

Значение	PICTURE	Машинная форма	Тип
6.92	X (4)	_____	нечисловое
6.92	_____	692	числовое
6.92	S999V99	0069B	_____
_____	VPP999	692	числовое
692000	_____	692	числовое
692	9(3)	692	_____
692	X(3)	692	_____

4. Напишите фразы PICTURE, определяющие объем внутренней памяти для хранения следующих значений:

- а) 1976 числовое
- б) 1976 нечисловое
- в) ROBERT нечисловое
- г) 35.007 числовое
- д) +35.007 числовое
- е) .000089 числовое
- ж) .000089 нечисловое
- з) 166789 числовое

2.5. Статьи-описания-записей

Общий формат статьи-описания-записи, рассматривавшийся до сих пор, был таков:

$$\left\{ \begin{array}{l} \text{номер-уровня} \end{array} \left\{ \begin{array}{l} \text{имя-данного} \\ \text{FILLER} \end{array} \right\} \left[\text{PICTURE IS строка-литер} \right] . \right\} \dots$$

со следующими ограничениями:

1. Самая объемлющая группа, описываемая в статье, должна иметь номер-уровня 01.

2. FILLER является необязательной возможностью и может употребляться только для элементарных данных.

3. Фраза PICTURE должна использоваться в элементарной статье и нигде больше.

Статьи-описания-записей используются для описания способа представления данных во внутренней памяти машины, будь то в файловой области или в области рабочей-памяти. Таким образом, эти статьи могут появляться как в секции FILE SECTION, так и в секции WORKING-STORAGE SECTION раздела данных. В секции WORKING-STORAGE SECTION статьи-описания-записей называются по-другому, а именно статьями-описания-данных. Имеется только одно различие между статьями-описания-записей и статьями-описания-данных, оно состоит в том, что, когда программа впервые загружается в память, элементарным данным в рабочей памяти могут быть присвоены начальные значения. Элементарным данным в области файлов значения могут присваиваться только с помощью операторов READ (ЧИТАТЬ) или MOVE (ПОМЕСТИТЬ), и этим данным нельзя присвоить начальных значений. Имея в виду это небольшое, но существенное различие, слова статья-описания-записи и статья-описания-данного можно употреблять с равным успехом. Полностью раздел данных определяется следующим образом:

- , заголовок-раздела-данных.
- заголовок-секции-файлов.
- статьи-описания-файлов и статьи-описания-записей.
- заголовок-секции-рабочей-памяти.
- статьи-описания-данных.

где все заголовки начинаются в поле A и являются следующими:

DATA DIVISION.

FILE SECTION.

WORKING-STORAGE SECTION.

Описание-файла состоит из индикатора-уровня-описания-файла (буквы FD (ОФ)) со следующими за ним именем-файла и одной (а позднее и более) описательной фразой. Каждая статья-описания-файла именует и описывает один из файлов, которые будут обрабатываться КОБОЛ-программой; с ней связана одна или более статья-описания-записи. До сих пор упоминалось о существовании только одного типа записей в данном файле, но позднее будет видно, что файл может содержать много различных типов записей. Общий формат для статьи-описания-файла таков:

$$\left[\begin{array}{l} \text{FD имя-файла} \\ \text{LABEL} \end{array} \left\{ \begin{array}{l} \text{RECORD IS} \\ \text{RECORDS ARE} \end{array} \right\} \left\{ \begin{array}{l} \text{STANDARD} \\ \text{OMITTED} \end{array} \right\} \right. \\ \left. \left\{ \begin{array}{l} \text{01 имя-данного-1} \\ \text{номер-уровня} \end{array} \right\} \left[\begin{array}{l} \text{PICTURE IS строка-литер-1} \\ \text{имя-данного-2} \\ \text{FILLER} \end{array} \right\} \left[\text{PICTURE IS строка-литер-2} \right] \dots \left\{ \dots \right\} \dots \right]$$

Примером полной секции файлов может служить следующая секция: FILE SECTION.

FD INVENTORY-INFORMATION-FILE

LABEL RECORDS ARE STANDARD.

01 MAIN-INVENTORY-DATA.

02 ITEM-CODE-X PICTURE IS X(6).

02 NUMBER-ON-HAND PICTURE IS 999.

02 COST-PER-UNIT PICTURE IS 9(4)V99.

FD ANOTHER-COLLECTION-OF-DATA

LABEL RECORD IS OMITTED.

01 FIRST-RECORD-TYPE PICTURE IS X(200).

01 SECOND-RECORD-TYPE.

05 NEW-ITEM-CODE PICTURE IS X(5).

05 DESCRIPTIVE-NAME-X PICTURE IS X(95).

Номера-уровней для всех данных, подчиненных групповому данному, представляющему собой запись, должны быть больше, чем 01, но имейте в виду, что они не обязаны быть последовательными, а должны только быть большими.

Заголовок-секции-файлов должен предшествовать заголовку-секции-рабочей-памяти, если они оба присутствуют в программе. Трудно вообразить КОБОЛ-программу без секции файлов или вообразить осмысленную программу без обеих секций. В программе

будет столько статей-описания-файлов, сколько файлов используется в программе. Каждая статья FD должна иметь соответствующую статью в параграфе FILE-CONTROL раздела оборудования. Формат описания секции рабочей-памяти таков:

WORKING-STORAGE SECTION.

```
[ 01 имя-данного-3 [ PICTURE IS строка-литер-3 ]
  [ номер-уровня [ имя-данного-4 ]
    [ FILLER ] [ PICTURE IS строка-литер-4 ]
    [ VALUE IS литерал ] . ] ... ] ...
```

В области рабочей памяти данное может принимать значение одним из двух способов: данное из какого-либо другого места памяти, будь то файловая область или область рабочей-памяти, может быть перемещено в рассматриваемое данное или рассматриваемое данное может иметь некоторое начальное значение, присвоенное программой при первоначальной загрузке для исполнения в память машины. Задание начальных значений осуществляется с помощью фразы VALUE IS литерал (ЗНАЧЕНИЕ литерал). Эта фраза указывает на значение, которое должно быть первоначально присвоено данному в рабочей-памяти. Это значение устанавливается только один раз при загрузке программы. Любая последующая обработка, за исключением пересылки из данного, будет изменять его значение.

Имеется два ограничения на использование фразы VALUE. Первое состоит в рекомендации начинающим использовать ее только на уровне элементарных данных, хотя фраза VALUE может использоваться и для групповых данных. Второе ограничение заключается в том, что фраза VALUE не должна противоречить фразе PICTURE. Если фраза PICTURE определяет нечисловое данное, литерал во фразе VALUE должен быть нечисловым, как, например:

01 SAMPLE-X PICTURE IS X(9) VALUE IS "PERSONNEL".

Число литер в нечисловом литерале не должно превышать размера данного. Если в нем меньше литер, чем определено, справа будут автоматически добавляться пробелы. Если фраза PICTURE определяет числовое данное, литерал во фразе VALUE должен быть числовым, и значение его должно находиться внутри диапазона значений, указанного фразой PICTURE. Например, для шаблона VPPP99 литерал должен быть в пределах от .00000 до .00099:

01 CONSTANT-VALUE PICTURE IS S999V9 VALUE IS —35.7.

В дополнение к литералам, упомянутым выше и описанным ранее, в КОБОЛе имеются величины, называемые *стандартными констан-*

тами, которые можно использовать как литералы. Эти зарезервированные слова не должны заключаться в кавычки. Все стандартные константы имеют нечисловые значения, за исключением константы ZERO (НУЛЬ), которая либо имеет числовое значение ноль, либо представляет собой строку литер 0. Каково именно ее значение, определяется контекстом, в котором она используется. Например, статья

01 CONSTANT-X PICTURE IS S99V9 VALUE IS ZERO.

задает присвоение начального значения 0.0 данному CONSTANT-X, в то время как статья

01 CONSTANT-Y PICTURE IS X(5) VALUE IS ZERO.

задает присвоение нечисловой строки 00000 данному CONSTANT-Y. Допускается употребление имен стандартных констант в виде слов в форме единственного или множественного числа; такие имена эквивалентны и взаимозаменяемы. Различная форма слов допускается для удобства использования и, быть может, для внесения некоторого дополнительного смысла в статью программы, но форма слова не влияет на присваиваемое значение.

Имена данных

Смысл

{ ZERO
ZEROS
ZEROES }
{ НУЛЬ
НУЛИ }

Представляет значение 0 или одну или более литер «0» в зависимости от контекста.

Пример

01 CONSTANT-Z PICTURE IS S9(10)V9(5) VALUE IS ZERO.

{ SPACE
SRACES }
{ ПРОБЕЛ
ПРОБЕЛЫ }

Представляет один или более (нужное количество) пробелов или пропусков.

Пример

01 OUTPUT-TITLE PICTURE IS X(35) VALUE IS SPACES.

{ HIGH-VALUE }
 { HIGH-VALUES }
 { НАИБОЛЬШЕЕ- }
 { ЗНАЧЕНИЕ }
 { НАИБОЛЬШИЕ- }
 { ЗНАЧЕНИЯ }

Представляет одну или более литер, которые имеют наибольшее значение для рабочей машины (на каждой машине все литеры упорядочены в определенной последовательности, служащей для целей сравнения).

Пример

01 MAXIMUM-LEVEL PICTURE IS X(5) VALUE IS HIGH-VALUE.

{ LOW-VALUE }
 { LOW-VALUES }
 { НАИМЕНЬШЕЕ- }
 { ЗНАЧЕНИЕ }
 { НАИМЕНЬШИЕ- }
 { ЗНАЧЕНИЯ }

Представляет одну или более литер, которые имеют наименьшее значение в сравнительной последовательности рабочей машины; это не обязательно литера 0 или A, она зависит от битовой кодировки.

Пример

01 SMALLEST-VALUE PICTURE IS X(5) VALUE IS LOW-VALUES.

{ QUOTE }
 { QUOTES }
 { КАВЫЧКА }
 { КАВЫЧКИ }

Представляет одну или более литер ", но слово QUOTE не может использоваться вместо знака кавычек в качестве ограничителя числового литерала.

Пример

01 STORY-LINE.
 05 SPECIAL-X

PICTURE IS X
 VALUE IS QUOTE.

05 TEXT-X

PICTURE IS X(9)
 VALUE IS "UNHAND ME".

05 SPECIAL-Y

PICTURE IS X
 VALUE IS QUOTE.

05 TEXT-Y

PICTURE IS X(10)
 VALUE IS ",SHE SAID."

ALL (BCE) литерал

Представляет строку литер из одного или нескольких вхождений указанного литерала. Литерал должен быть либо нечисловым литералом, либо стандартной константой. Если он является стандартной константой, то слово ALL избыточно и используется только для удобства чтения.

Пример

```
01 ALPHA-X    PICTURE IS X(5)  VALUE IS ALL "A".
01 BETA-X     PICTURE IS X(5)  VALUE IS ALL SPACES.
01 GAMMA-X    PICTURE IS X(5)  VALUE IS ALL "AB".
```

В последнем примере начальным значением данного GAMMA-X будет ABABA. Применение стандартных констант не ограничивается фразой VALUE. Они могут использоваться в любом месте КОБОЛ-программы, где может использоваться нечисловой литерал (кроме константы ZERO, которая может использоваться и как числовая), например в операторе MOVE:

```
MOVE SPACES TO DATA-ITEM-X.
```

2.6. Примеры описаний данных

А. Рис. 1.10 в предыдущей главе показывает разметку бланка для записи программы на КОБОЛе. Напишите описание записи для этого стандартного расположения данных.

Б. Составьте запись для хранения информации в файле, используемом для напоминания людям важных дат и содержащем имя, адрес, номер телефона, а также место для пяти важных дат. Покажите примеры возможных записей.

В. Записи файла определяются следующим описанием:

```
01 STUDENT-PERFORMANCE-RECORD.
   05 STUDENT-NUMBER             PICTURE IS 9 (9).
   05 STUDENT-GRADES.
```


10	COURSE-NUMBER-1	PICTURE IS 9(5).
10	LETTER-GRADE-1	PICTURE IS X.
10	COURSE-NUMBER-2	PICTURE IS 9(5).
10	LETTER-GRADE-2	PICTURE IS X.
05	FILLER	PICTURE IS X(5).
05	DATES-OF-INTEREST.	
10	ENTRY-DATE.	
15	ENTRY-MONTH	PICTURE IS X(3).
15	ENTRY-DAY	PICTURE IS X(2).
15	ENTRY-YEAR	PICTURE IS X(4).
10	BIRTH-DATE.	
15	BIRTH-MONTH	PICTURE IS X(3).
15	BIRTH-DAY	PICTURE IS X(2).
15	BIRTH-YEAR	PICTURE IS X(4).

Приведите пример записи, отвечающей этому описанию. Заметьте, что она не может быть перфокартой, так как общая длина меньше, чем восемьдесят литер.

Г. Планируется прочитать образ 80-колоной карты, присоединить к нему справа данное в 7 колонок, скажем с именем SPECIAL, и напечатать получившуюся запись. Напишите раздел DATA DIVISION для такой программы, включающий секцию FILE SECTION, статьи FD и секцию WORKING-STORAGE SECTION.

Предлагаемые решения

A.

01	COBOL-PROGRAMMING-FORM.	
05	SEQUENCE-NUMBER	PICTURE IS 9(6).
05	CONTINUATION-X	PICTURE IS X.
05	TEXT-MATERIAL.	
10	AREA-A	PICTURE IS X(4).
10	AREA-B	PICTURE IS X(61).
05	IDENTIFICATION-ITEM	PICTURE IS X(20).

B.

01	REMINDER-RECORD.	
05	ADDRESS-ITEM.	
10	PERSONS-NAME	PICTURE IS X(25).
10	STREET-ADDRESS	PICTURE IS X(25).
10	CITY-STATE-AREA.	

	15 CITY-NAME	PICTURE IS X(20).
	15 STATE-NAME	PICTURE IS X(2).
	15 ZIP-CODE	PICTURE IS X(5).
05	PHONE-NUMBER	PICTURE IS X(7).
05	IMPORTANT-DATES.	
	10 DATE-ONE	PICTURE IS 9(6).
	10 DATE-TWO	PICTURE IS 9(6).
	10 DATE-THREE	PICTURE IS 9(6).
	10 DATE-FOUR	PICTURE IS 9(6).
	10 DATE-FIVE	PICTURE IS 9(6).
	10 DATE-SIX	PICTURE IS 9(6).

B.

982394685I0235A11155B DEC30I972MAY031955

Г.

DATA DIVISION.

FILE SECTION.

FD CARD-IMAGE-FILE

LABEL RECORD IS STANDARD.

01 CARD-IMAGE-RECORD PICTURE IS X(80).

FD SPECIAL-IMAGE-FILE

LABEL RECORD IS STANDARD.

01 SPECIAL-IMAGE-RECORD.

05 INPUT-RECORD-IMAGE PICTURE IS X(80).

05 PLACE-FOR-WORD-SPECIAL PICTURE IS X(7).

WORKING-STORAGE SECTION.

01 WORD-SPECIAL PICTURE IS X(7) VALUE IS
"SPECIAL".

Упражнения

I. Необходимо приготовить файл, содержащий записи, относящиеся к имени и адресу подписчиков, получающих журналы по почте. Каждая запись должна содержать:

имя подписчика

название улицы, где живет подписчик

город, штат, почтовый индекс подписчика

срок истечения подписки

сумму по последнему счету

учетный процент от стоимости подписки

Напишите статью-описания-записи для такого файла. Сделайте разумные предположения относительно длины и типа каждого данно-

го. Составьте запись таким образом, чтобы адресуемая информация могла перемещаться как единое данное. Учетный процент должен быть дробным и будет впоследствии умножаться на стоимость подписки для определения нового счета.

2. Напишите два примера записей, описанных следующими статьями-описания-записей:

01 EXAMPLE-RECORD.

05	VENDOR-NUMBER	PICTURE IS 9(5).
05	VENDOR-NAME	PICTURE IS X(15).
05	INVOICE-AMOUNT	PICTURE IS 9(4)V99.
05	DUE-DATES-X.	
10	INVOICE-DATE	PICTURE IS 9(6).
10	DISCOUNT-DATE	PICTURE IS 9(6).
10	NET-DUE-DATE	PICTURE IS 9(6).

3. Напишите описание записи, имеющей длину восемьдесят литер, с которой можно обращаться, как с одним 80-литерным данным или с двумя 40-литерными данными, или с десятью 8-литерными данными в зависимости от требований раздела процедур. Имейте в виду, что должно быть десять 8-литерных данных, а не восемь 10-литерных.

2.7. Глоссарий

Групповое данное. Поименованный набор смежных элементарных и групповых данных.

Данное. Любое элементарное данное, поименованная группа элементарных данных в составе записи, поименованная группа групповых данных или сама запись.

Зарезервированное слово. Одно из точно установленного списка слов, которые могут использоваться в исходных КОБОЛ-программах, но которые не должны появляться в программе как слова, определенные пользователем.

Знак числа. Алгебраический знак, связанный с числовым данным и указывающий положительное оно или отрицательное; часто включается в позицию последней литеры данного.

Логическая запись. Наиболее объемлющее данное.

Нечисловое данное. Данное, описанное таким образом, что его значение может быть любой комбинацией литер из набора буквенно-цифровых литер.

Номер уровня. Две цифровые литеры, указывающие иерархическую структуру логической записи.

Подразумеваемая десятичная точка. Позиция десятичной точки, которой не соответствует в данном явная литера точки.

Подразумеваемая десятичная точка имеет арифметический смысл, но не имеет физического представления.

Последовательный доступ. Метод доступа к файлу, при котором у логической записи, извлекаемой из файла или размещаемой в нем, имеется логический предшественник и логический преемник. При первом обращении к файлу доступной становится запись, не имеющая предшественника; каждое последующее обращение ссылается на преемника логической записи, к которой было предыдущее обращение.

Присоединить. Связать вместе в последовательность или цепочку; сформировать смежную последовательность.

Связанные данные. Данные, описанные посредством последовательных статей раздела данных и имеющие определенную взаимосвязь друг с другом.

Стандартная константа. Зарезервированное слово, представляющее числовое значение, литеру или строку литер.

Статья описания данного (или записи). Статья в разделе данных, состоящая из номера уровня, за которым следует имя данного, а затем, если требуется, набор фраз данных.

Статья описания файла. Статья в секции файлов раздела данных, состоящая из индикатора уровня FD (ОФ), за которым следует имя-файла, а затем набор необходимых файловых фраз.

Строка литер. Последовательность смежных литер КОБОЛа, образующая литерал, слово или строку-литер шаблона в разделе данных.

Файл. Совокупность логических записей, которой обычно предшествуют (и за которой следуют) записи меток.

Числовое данное. Данное, описание которого ограничивает представление его значения литерами цифр, оно может содержать либо нет знак числа.

Элементарное данное. Данное, которое описывается как далее логически неделимое. Элементарное данное должно иметь в статье-описания-данного фразу PICTURE (ШАБЛОН).

GO TO CERTAIN-PARAGRAPH

вызывает изменение этого порядка выполнения и передает управление первому оператору параграфа с именем CERTAIN-PARAGRAPH.

Предложения состояются из одного или более операторов, как в примере:

```
ADD 1.0 TO SUM-X MOVE SUM-X TO RESULT-X  
GO TO LAST-PARAGRAPH.
```

Каждый оператор КОБОЛа начинается с глагола, за которым следует одна или несколько фраз. В приведенном выше примере фигурировали глаголы: ADD, MOVE и GO. Чаще всего предложение составляется из единственного оператора, и та же самая последовательность действий могла бы быть выполнена с помощью следующих предложений:

```
ADD 1.0 TO SUM-X.  
MOVE SUM-X TO RESULT-X.  
GO TO LAST-PARAGRAPH.
```

Последняя форма предпочтительнее так как, чем проще структура программы, тем меньше возможностей для ошибок. Хотя в одной строке можно записывать несколько предложений, привычка писать только по одному предложению в строке облегчает внесение исправлений. В реальном программировании первоначальный текст программы никогда не бывает окончательным, так как со временем возникает необходимость вносить в него изменения. Имеются, конечно, случаи, когда бывают необходимы составные предложения, например:

```
READ INPUT-FILE RECORD  
AT END MOVE "FINISHED" TO OUTPUT-RECORD  
ADD 1.0 TO COUNTER-X2  
CLOSE INPUT-FILE  
GO TO END-OF-PROGRAM.
```

Единственное, чем при написании отличается предложение, состоящее из оператора, от оператора — это завершающая точка, расположенная в конце.

В базовом КОБОЛе имеется только четырнадцать глаголов, и каждый из них идентифицирует определенный тип оператора. Эти четырнадцать глаголов можно сгруппировать в три класса:

1. Операторы обмена данными:

CLOSE	(ЗАКРЫТЬ)
DISPLAY	(ВЫДАТЬ)
MOVE	(ПОМЕСТИТЬ)
OPEN	(ОТКРЫТЬ)

READ	(ЧИТАТЬ)
WRITE	(ПИСАТЬ)

2. Арифметические операторы:

ADD	(СЛОЖИТЬ)
COMPUTE	(ВЫЧИСЛИТЬ)
DIVIDE	(РАЗДЕЛИТЬ)
MULTIPLY	(УМНОЖИТЬ)
SUBTRACT	(ОТНЯТЬ)

3. Операторы управления:

GO	(ПЕРЕЙТИ)
IF	(ЕСЛИ)
STOP	(ОСТАНОВИТЬ)

Различные типы операторов будут описаны в этой и следующих двух главах. После их изучения читатель может считать, что он знаком с базовым КОБОЛом. Хотя в полном языке имеются еще и другие глаголы, в современном коммерческом программировании в 95% случаев используются именно эти четырнадцать глаголов. Начиная с гл. 6, будут рассмотрены другие глаголы и приведены примеры их употребления.

Ниже представлен пример раздела процедур сначала без секций, а затем с заголовками-секций. Обратите внимание, что имена-параграфов начинаются с букв, что является обязательным для всех имен КОБОЛа. Хотя допустимо иметь полностью числовые имена-параграфов, это не рекомендуется. Все имена-параграфов начинаются в поле А так же, как и имена-секций, и заголовков раздела. Все предложения начинаются в поле В.

Пример раздела процедур без секций

PROCEDURE DIVISION.

INITIALIZE-X.

OPEN INPUT NEW-ENTRY-FILE MASTER-FILE
OUTPUT NEW-MASTER-FILE.

READ-NEW-ENTRY-FILE.

READ NEW-ENTRY-FILE RECORD AT END
GO TO COPY-FILE.

READ-MASTER-FILE.

READ MASTER-FILE RECORD AT END
CLOSE NEW-ENTRY-FILE MASTER-FILE.
NEW-MASTER-FILE
STOP RUN.

IF NEW-STOCK-NUMBER IS EQUAL TO STOCK-NUMBER
GO TO REPLACE-RECORD.

MOVE MASTER-RECORD TO NEW-MASTER-RECORD.

WRITE NEW-MASTER-RECORD.

GO TO READ-NEW-ENTRY-FILE.

REPLACE-RECORD.

MOVE NEW-ENTRY-RECORD TO NEW-MASTER-RECORD.

WRITE NEW-MASTER-RECORD.

COPY-FILE.

MOVE LOW-VALUES TO NEW-STOCK-NUMBER.

GO TO READ-MASTER-FILE.

Пример раздела процедур с использованием секций

PROCEDURE DIVISION.

READ-FILES SECTION.

INITIALIZE-X.

OPEN INPUT NEW-ENTRY-FILE MASTER-FILE

OUTPUT NEW-MASTER-FILE.

READ-NEW-ENTRY-FILE.

READ NEW-ENTRY-FILE RECORD AT END

GO TO COMPLETE-RUN.

READ-MASTER-FILE.

READ MASTER-FILE RECORD AT END CLOSE

NEW-ENTRY-FILE MASTER-FILE

NEW-MASTER-FILE STOP RUN.

IF NEW-STOCK-NUMBER IS EQUAL TO STOCK-NUMBER

GO TO UPDATE-MASTER.

MOVE MASTER-RECORD TO NEW-MASTER-RECORD.

WRITE NEW-MASTER-RECORD.

GO TO READ-MASTER-FILE.

UPDATE-MASTER SECTION.

REPLACE-RECORD.

MOVE NEW-ENTRY-RECORD TO NEW-MASTER-RECORD.

WRITE NEW-MASTER-RECORD.

GO TO READ-NEW-ENTRY-FILE.

COMPLETE-RUN SECTION.

COPY-FILE.

MOVE LOW-VALUES TO NEW-STOCK-NUMBER.

GO TO READ-MASTER-FILE.

3.2. Операторы OPEN-CLOSE

Каждый файл, подлежащий обработке с помощью КОБОЛ-программы, должен быть открыт прежде, чем его записи можно будет считывать во внутреннюю память. Каждый файл, который был открыт для чтения, должен быть также закрыт программой после завершения его обработки.

Оператор OPEN (ОТКРЫТЬ) достаточно прост для написания программистами, но выполняет многочисленные действия. Общий формат оператора OPEN таков:

$$\text{OPEN} \left\{ \begin{array}{l} \text{INPUT} \quad \{\text{имя-файла-1}\} \dots \\ \text{OUTPUT} \{\text{имя-файла-2}\} \dots \end{array} \right\} \dots$$

В операторе OPEN каждая из возможностей (INPUT (ВХОДНОЙ) или OUTPUT (ВЫХОДНОЙ)) для одного файла может быть указана только один раз. Тем не менее обе из них (для разных файлов) могут встречаться в одном операторе OPEN и в любом порядке. В одном операторе OPEN можно открыть любое нужное количество файлов, и в простых программах обычно одним оператором открывается несколько файлов, например:

```
OPEN INPUT CARD-IMAGE-FILE OUTRUT ERROR-FILE
      OUTPUT-RECORD-FILE PRINTER-FILE.
```

Обратите внимание, что в этом примере один файл открыт как входной, а три файла открыты как выходные. Эти же действия могли бы быть заданы другим способом:

```
OPEN INPUT CARD-IMAGE-FILE.
OPEN OUTPUT ERROR-FILE.
OPEN OUTPUT OUTPUT-RECORD-FILE.
OPEN OUTPUT PRINTER-FILE.
```

В сложных программах различные файлы могут открываться и закрываться во многих точках программы.

Оператор OPEN не читает и не выбирает первую запись данных и не записывает никакой записи данных. Для передачи записей из внешней памяти во внутреннюю и наоборот служат операторы READ и WRITE, которые могут выполняться только после открытия файлов с помощью операторов OPEN. Оператор OPEN должен быть выполнен для любого файла прежде, чем для этого файла будут выполняться операторы READ или WRITE. Для файла нужно выполнить только один оператор OPEN; файл остается открытым для всех последующих операций ввода и вывода до тех пор, пока не встретится оператор CLOSE (ЗАКРЫТЬ). Нельзя выполнять второй оператор OPEN для файла прежде, чем для него будет выполнен оператор CLOSE,

Для того чтобы понять назначение операторов OPEN и CLOSE, необходимо отчетливо представлять физическую сущность вычислительной системы. Файл — это совокупность записей, которая

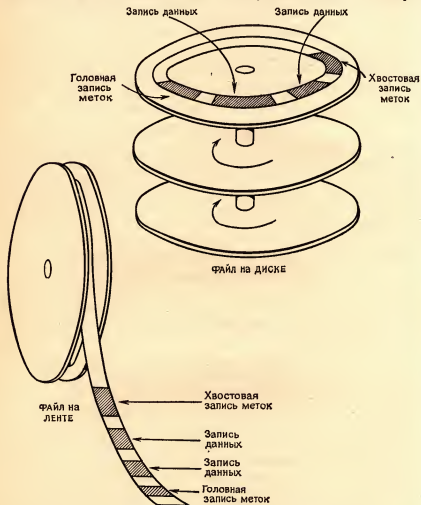


Рис. 3.1. Пример файлов, показывающий организацию данных.

помещается на некоторый носитель, например на магнитную ленту, пакет дисков или колоду перфокарт. Эти записи представляют собой логические записи, формат которых описывается с помощью статей-описания-записи в разделе данных. В добавление к этим записям данных в начале и конце их совокупности помещаются специальные

вспомогательные записи меток. На эти записи меток ссылаются, в частности, с помощью фразы LABEL RECORDS ARE STANDARD (МЕТКИ СТАНДАРТНЫ) в статье-описания-файла. При работе с файлами на перфокартах эти записи меток иногда могут быть опущены, но с появлением вычислительных систем, имеющих много

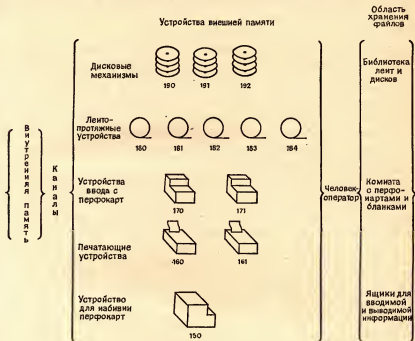


Рис. 3.2. Различные типы устройств для установки носителей файлов.

устройств чтения перфокарт, все чаще и чаще такие файлы также снабжаются метками. Записи меток не содержат данных для программы, а содержат данные, описывающие файл: дату создания, число логических записей в нем и т. д. На рис. 3.1 показана последовательность записей, образующая файл.

Носитель файла (лента, диск или колода перфокарт) устанавливается на некоторое физическое устройство, например ленто-протяжное устройство, дисковый механизм или устройство ввода перфокарт. Как показано на рис. 3.2, может быть много устройств каждого типа. Обычно дисковые механизмы имеют съемные пакеты дисков, хотя существуют и такие, на которых пакеты дисков не снимаются, а установлены раз и навсегда. Катушки на всех лентопротяжных устройствах сменяемы. Всегда сменяемы колоды перфокарт и перфоленты. Все установки и снятия сменных носителей произ-

водятся вручную оператором. Он подготавливает колоды карт и перфоленту, доставляет катушки магнитной ленты из библиотеки и отправляет ленты с результатами в отведенное для них место. Необходимость всех этих действий должна учитываться программистом. При этом должен быть предусмотрен контроль правильности их выполнения. Часть такого контроля осуществляется с помощью операторов OPEN и CLOSE.

Предусмотрено три типа имен для файла, для носителя и для реального устройства: имена-файлов, имена-катушек/пакетов и имена-устройств. *Имя-файла* используется в программе на КОБОЛе для идентификации совокупности записей. *Имя-катушки/пакета* записывается на катушку магнитной ленты или пакет дисков для идентификации носителя или его части, содержащей файл. В терминологии операционной системы IBM оно называется «имя-набора-данных». *Имя-устройства* идентифицирует тип устройства, на котором может находиться катушка или пакет дисков.

Может показаться, что проще иметь только одно имя и обходиться им. Но наличие одного имени не обеспечило бы гибкость, необходимую для современных применений вычислительных машин. Неизвестно, какое из устройств на данной установке будет доступно для использования во время выполнения программы. Было бы неудобно, если бы пришлось ждать, пока освободится конкретное устройство. Гораздо лучше иметь возможность использовать любое устройство и сообщать операционной системе только его тип. Информация на катушке или пакете дисков может быть записана в другом вычислительном центре, а их имена могут отличаться от имен, используемых конкретным программистом. На рис. 3.3 приведен пример взаимосвязи указанных трех типов имен. Имени-файла CARD-IMAGE-FILE соответствует имя AM398, занесенное в запись метки катушки. В текущий момент эта катушка находится на устройстве с номером 183 (на одном из многих лентопротяжных устройств данного вычислительного центра). Связь между именем-файла и именем-устройства устанавливается с помощью фразы SELECT (ВЫБИРАЯ) с учетом выдаваемого оператором сообщения, указывающего имя-устройства (на некоторых установках это делается с помощью управляющих карт языка управления заданиями). Связь между именем-файла и именем-катушки устанавливается с помощью фразы VALUE OF (ЗНАЧЕНИЕ) или с помощью управляющих карт операционной системы.

Когда в рабочей КОБОЛ-программе выполняется оператор:

OPEN INPUT имя-файла

операционная система будет искать файл, имя которого указано в операторе OPEN. Если он не может быть найден, то распечатывается или выводится на дисплей соответствующее сообщение оператору машины, который установит нужный файл на некоторое физи-

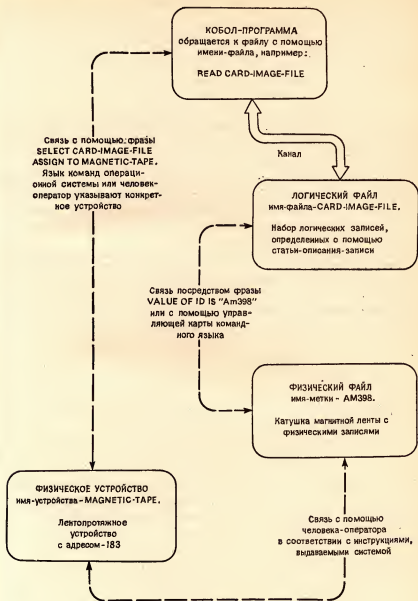


Рис. 3.3. Связь имен.

ческое устройство и проинформирует об этом систему. При наличии фразы LABEL RECORDS ARE STANDARD (МЕТКИ СТАНДАРТ-НЫ) по оператору OPEN будут прочитаны записи меток файла и будет проверена их приемлемость. Если метки правильны, то файл устанавливается в такое положение, что последующий оператор READ введет его запись. Когда выполняется оператор:

OPEN OUTPUT имя-файла

имеет место точно такой же поиск требуемой катушки ленты или пакета дисков с именем-файла и проверяются записи меток. Если метки выходного файла приемлемы, то записи меток корректируются с учетом новой информации (текущей даты, времени хранения и т. д.), которая служит признаком нового файла. После этого выходной файл устанавливается для выполнения последующего оператора WRITE. Нельзя применять оператор READ к файлу, который был открыт как OUTPUT (ВЫХОДНОЙ) файл. INPUT файл можно использовать только для ввода, а OUTPUT файл можно использовать только для вывода. Один и тот же файл нельзя использовать и для того, и для другого одновременно. При вводе файл не разрушается и оператор CLOSE, за которым следует другой оператор OPEN, вернет файл к началу для повторения ввода записей.

Оператор CLOSE должен использоваться в исходной КОБОЛ-программе, когда программа заканчивает работу с данным файлом. Общий формат оператора CLOSE прост:

CLOSE {имя-файла} ...

Обратите внимание, что здесь нет необходимости в словах INPUT или OUTPUT. Оператор закрывает и те, и другие файлы. И снова в одном операторе можно указать столько файлов, сколько необходимо программисту, например:

CLOSE CARD-IMAGE-FILE ERROR-RECORD-FILE
OUT-RECORD-FILE PRINTER-FILE.

После того как для некоторого файла выполнен оператор CLOSE, операторы READ и WRITE для этого файла не должны выполняться до тех пор, пока не будет выполнен новый оператор OPEN. Ниже дан пример правильной последовательности этих операторов:

PROCEDURE DIVISION.
PARAGRAPH-ONE.

OPEN INPUT CARD-IMAGE-FILE OUTPUT
PRINTER-FILE.

READ CARD-IMAGE-FILE-RECORD AT END GO TO
PARAGRAPH-TWO.

MOVE CARD-IMAGE-RECORD TO PRINTER-RECORD.

WRITE PRINTER-RECORD.
PARAGRAPH-TWO.
CLOSE CARD-IMAGE-FILE PRINTER-FILE.
STOP RUN.

Выполнение оператора CLOSE приведет к занесению записей хвостовых меток, если файл был открыт ранее как OUTPUT файл. Если же файл был открыт ранее как INPUT файл, то хвостовые метки не проверяются. В обоих случаях файл возвращается в исходное состояние; это аналогично перемотке к началу катушки магнитной ленты. Закрытие файла, связанного с устройством чтения карт или устройством построочной печати, не приведет к возврату файла в исходное состояние.

Упражнения

1. Составьте блок-схему раздела процедур из предыдущего раздела книги, использующего имена секций.
2. В следующем примере процедуры некоторые операторы написаны неправильно, в то время как другие записаны в неправильной последовательности. Среди них даже есть оператор без единой ошибки. Опишите каждый оператор с точки зрения его правильности (операторы помечены для ссылок буквами).

EXAMPLE-PARAGRAPH.

- а. OPEN INPUT-FILE.
 - б. OPEN OUTPUT TAPE-FILE-ZULU.
 - в. CLOSE TAPE-FILE-THREE TAPE-FILE-ZULU.
 - г. OPEN INPUT TAPE-FILE-ZULU.
 - д. OPEN OUTPUT TAPE-FILE-SECOND INPUT
TAPE-FILE-TEST TAPE-FILE-ZULU.
3. Ответьте на следующие вопросы:
- а) Как должны размещаться в полях части стандартного бланка раздела процедур?
 - б) Сколько раз может быть открыт отдельный файл?
 - в) Каково определение предложения?

3.3. Операторы READ-WRITE

После того как файл открыт в качестве INPUT файла, можно выполнять оператор READ. Всякий раз при выполнении оператора READ одна логическая запись считывается в файловую область

внутренней памяти, а файл устанавливается на следующую запись. Общий формат оператора READ таков:

READ имя-файла RECORD AT END повелительное-предложение

Заметьте, что, хотя здесь есть необязательные слова (RECORD и AT), которые можно опустить, оператор READ не имеет вариантов. Фраза AT END должна быть включена. Даже если программист уверен, что в его программе конец файла не будет достигнут, он все же обязан вставить фразу AT END. Примеры операторов READ:

```
READ CARD-INPUT-FILE RECORD AT END STOP RUN.  
READ TRANSACTION-FILE RECORD AT END  
GO TO ERROR-STEP.  
READ FILE-THIRTY-TWO RECORD AT END  
MOVE "FINISH" TO OUTPUT-RECORD  
ADD 4.3 TO COUNTER-ENTRY  
WRITE OUTPUT-RECORD  
STOP RUN.
```

Запомните: прежде чем может быть выполнен оператор READ, должен быть выполнен оператор OPEN INPUT имя-файла, т. е. соответствующий файл должен быть открыт как входной.

При каждом выполнении оператора READ (будь то в результате наличия в программе нескольких таких операторов или в результате повторных передач управления тому же самому оператору) только одна запись передается в область, определенную в статье-описания-записи раздела данных. В свою очередь содержимое этой области заменяется каждой новой считываемой записью. Если значение записи нужно сохранить от этого разрушающего ввода, то запись следует переслать в другую область перед выполнением следующего оператора READ. Если после считывания последней записи выполняется еще один оператор READ, то выполняется фраза AT END (В КОНЦЕ). Содержимое области записи при этом оказывается неопределенным, и выполняются повелительные-операторы. Эти повелительные-операторы могут представлять собой либо единственный оператор, как например STOP RUN или GO TO END-OF-FILE-PARAGRAPH, либо длинную последовательность подряд расположенных операторов. Такая последовательность должна заканчиваться точкой и пробелом с тем, чтобы отделить эту последовательность от операторов, следующих непосредственно за оператором READ. Повелительные-операторы во фразе AT END действительно должны быть повелительными; они не могут быть условными операторами, допускающими выбор или принятие решения.

Оператор WRITE (ПИСАТЬ) передает запись из файловой области в некоторый выходной файл. Его формальное определение

таково:

WRITE имя-записи [FROM имя-данного]

Обратите внимание, что в операторе READ упоминается имя-файла, а в операторе WRITE — имя-записи. Имя-записи — это имя, используемое в статье-описания-записи статьи FD (ОФ) выходного файла. Прежде чем может быть выполнен оператор WRITE, для соответствующего файла должен быть выполнен оператор OPEN OUTPUT с соответствующим именем-файла. После того как запись, идентифицируемая именем-записи, переписана в файл, ее значение становится не доступным более программе. Оно фактически разрушается. Программист должен был либо закончить работу с этой записью, либо переписать ее куда-нибудь до выполнения оператора WRITE. Таким образом, области записей в файловой области являются как бы временной памятью: их содержимое разрушается операторами READ и WRITE. Примеры операторов WRITE:

WRITE OUTPUT-RECORD

WRITE RECORD-OF-RESULTS FROM INPUT-RECORD

При использовании варианта со словом FROM (ИЗ) перед выводом происходит перемещение записи данных из области, определенной именем-данного, в область, идентифицируемую именем-записи. Делается то же самое, что при выполнении двух операторов, первый из которых — это оператор пересылки MOVE (ПОМЕСТИТЬ):

MOVE INPUT-RECORD TO RECORD-OF-RESULTS

WRITE RECORD-OF-RESULTS

Имя-данного и имя-записи не должны совпадать, так что следующий оператор недопустим:

WRITE OUTPUT-RECORD FROM OUTPUT-RECORD

(НЕВЕРНО!)

Полное описание оператора MOVE будет дано позднее; говоря кратко, его действие состоит в пересылке из одного участка внутренней памяти в другой, будь то в файловой области или в области рабочей памяти.

3.4. Операции перемещения

В языке программирования, разработанном для коммерческих приложений, переорганизация данных, возможно, более важна, чем арифметические операции. Большинство из обрабатываемых данных относится к нечисловым данным, как, например, имена людей, да и обработка тех данных, которые являются числовыми, очень часто

сводится к простому сложению. Результатом обработки файлов обычно является отчет, предназначенный для чтения человеком. Организация информации в этом отчете может влиять на легкость его понимания читателем.

Следовательно, оператор MOVE (ПОМЕСТИТЬ) со всеми его разновидностями занимает важное место в программировании. Общий формат оператора MOVE таков:

$$\text{MOVE } \left\{ \begin{array}{l} \text{имя-данного-1} \\ \text{литерал} \end{array} \right\} \text{ TO } \text{имя-данного-2} [\text{имя-данного-3}] \dots$$

Имя-данного-1 или литерал задает пересылаемую область, а имя-данного-2 (и имя-данного-3 и т. д.) определяет принимающую область. Ниже приводятся примеры правильных операторов MOVE:

MOVE "NAMING" TO OUTPUT-ITEM-FIVE

MOVE ZERO TO ITEM-ONE FLAG-V RECORD-ZONE

MOVE INPUT-RECORD TO OUTPUT-RECORD

Действие любого оператора MOVE, в котором как пересылаемое, так и принимающее данное являются элементарными данными, называется элементарным перемещением. Элементарное данное можно просто распознать по наличию фразы PICTURE в статье-описания-данного. Поэтому, так как оба данных

01 RECORD-A PICTURE IS X (20).

и

01 RECORD-B PICTURE IS X (20).

являются элементарными, следующий оператор задает элементарное перемещение:

MOVE RECORD-A TO RECORD-B.

Когда оба данных являются нечисловыми и их размеры различны, значение пересылаемого данного копируется в принимающее данное, начиная слева. Если пересылаемое данное короче принимающего данного, то справа добавляются пробелы; если же пересылаемое данное длиннее принимающего, то значение усекается на правом краю принимающего данного, например:

Значение пересылаемого данного	Шаблон принимающего данного	Значение принимающего данного
A B C D E	X(4)	A B C D
A B C D E	X(5)	A B C D E
A B C D E	X(6)	A B C D E пробел

Когда оба данных числовые, происходит выравнивание относительно десятичной точки и необходимое дополнение нулями в соответствии с шаблоном принимающего данного. Если число цифр пересылаемого данного слева или справа от подразумеваемой десятичной точки больше, чем указано в шаблоне принимающего данного, то «лишние» цифры будут потеряны. Алгебраический знак в этом процессе выравнивания не учитывается. Если в шаблоне принимающего данного отсутствует знак числа (S), то передается абсолютная величина пересылаемого данного: Например, если бы значение пересылаемого данного было —832.56, то значения принимающего данного для следующих шаблонов были бы таковы:

Шаблон принимающего данного	Значение принимающего данного
999V99	83256
S99V99	—3256
S9999V	—0832
SV9999	—5600
S9V999	—2560
S9(5)V9(3)	—00832560

Таким образом, из сказанного выше следует сделать вывод, что нужно проявлять некоторую осторожность с тем, чтобы не изменить значений данных, перемещаемых в памяти.

Если в операторе MOVE одно или оба данных не являются элементарными, то такой оператор задает групповое перемещение. При групповом перемещении вся передача рассматривается как нечисловая, и вся заданная группа передается как одно данное. Групповое перемещение удобно, но при этом вновь возникает опасность усечения, если размер пересылаемого данного больше, чем размер принимающего данного. Если бы пересылаемое групповое данное было

01 FULL-GROUP.

05 NAME-X PICTURE IS X(5).

05 NUMBER-X PICTURE IS X(3).

а принимающее элементарное данное —

01 RECORD-A PICTURE IS X(6).

то в результате выполнения оператора

MOVE FULL-GROUP TO RECORD-A

две последние литеры данного NUMBER-X были бы потеряны.

Пример

В следующей таблице приведены результаты выполнения оператора.

MOVE SENDING-ITEM TO RECEIVING-ITEM.
(ПОМЕСТИТЬ ПЕРЕСЫЛАЕМОЕ В ПРИНИМАЮЩЕЕ.)

для различных значений величин и различных их шаблонов:

Пересылаемое данное			Принимающее данное		
Фактически хранящиеся литеры	Значение данного	PICTURE		Фактически хранящиеся литеры	Значение данного
12345	12,345.	9(5)	9(3)	345	345.
			9(5)V9(2)	1234500	12,345.00
			S9(5)	12345 ⁺	+12,345.
			9V99	500	5.00
6789	-67.89	S99V99	99V99	6789	67.89
			S9(3)V9(3)	067890	-67.890
			9(4)	0067	67.
			99	67	67.
ABCD	"ABCD"	X(4)	X(4)	ABCD	"ABCD"
			X(3)	ABC	"ABC"
			X	A	"A"
			X(5)	ABCD пробел	"ABCD "
1234	.001234	VPP9(4)	9(3)	000	0.
			V9(3)	001	.001
			V9(6)	001234	.001234
			9(3)V9(5)	00000123	.00123

Упражнения

1. Напишите полную КОБОЛ-программу, включающую все четыре раздела, которая осуществляет слияние записей двух отдельных входных файлов в третий выходной файл. Обе входных записи имеют одинаковый формат и описание PICTURE IS X(150). Прочтите запись из первого входного файла и запишите ее в выходной файл; прочтите запись из второго входного файла и запишите ее в выходной файл. Продолжайте попеременное чтение из каждого входного файла до тех пор, пока все записи не будут скопированы в выходной файл. В каждом из входных файлов будет одинаковое число записей и, следовательно, в выходном файле их будет в два раза больше.

Если в файле 1 имеется 100 записей, то и в файле 2 будет содержаться также 100 записей. В конце работы в файле 3 будет 200 записей. Предположим, что и исходная, и рабочая машины называются MARK-1, и поместим каждый файл на устройство, именуемое MAGNETIC-TAPE. Вы можете опустить фразу VALUE OF ID, если она не требуется на вашей вычислительной установке.

2. Заполните пропуски в следующей таблице:

Пересылаемое данное			Принимающее данное		
Фактическое значение	Хранящееся значение	PICTURE	Фактическое значение	Хранящееся значение	PICTURE
25	025	9(3)	25		99V9
55		999V9	55		99V9
-3.3	03L	S99V9			999
355	0355	9(4)	"0355"		X(4)
"1234"	1234	X(4)	1234		9(4)
	55	99PV			9(3)V9
1234	1234	9(4)			9(3)
"1234"	1234	X(4)			X(3)
123	1230	9(3)V9	123	12C	
12.34	1234	9(2)V99			9V9

3. Какие символы может содержать строка-литер во фразе PICTURE для описания числового данного?

4. Выпишите последовательность литер, в виде которой хранится в памяти следующая запись:

WORKING-STORAGE SECTION.

01 EXAMPLE-RECORD.

02 ALPHA-ITEM PICTURE IS X(5)
VALUE IS "ABCDE".

02 NUMERIC-PART.

03 A-ITEM PICTURE IS 999
VALUE IS 15.

03 B-ITEM PICTURE IS S99V99
VALUE IS -4.5.

03 C-ITEM PICTURE IS VPPP999
VALUE IS .000123.

02 END-VALUE PICTURE IS X
VALUE IS "\$".

3.5. Примеры заданий на программирование и программы

Пример А

Предположим, что записи файла INVENTORY-FILE имеют следующую статью-описания-записи:

```
01 INVENTORY-RECORD.
  05 ITEM-NUMBER.
    10 PLANT-IDENTIFICATION      PICTURE IS X(2).
    10 SEQUENCE-NUMBER           PICTURE IS X(6).
    05 ALPHABETIC-DESCRIPTION    PICTURE IS X(100).
    05 NUMBER-ON-HAND            PICTURE IS X(6).
    05 SUPPLIER-CODE.
      10 SUPPLIER-CLASS          PICTURE IS X(3).
      10 SUPPLIER-NUMBER         PICTURE IS X(4).
```

Напишите раздел процедур, который будет считывать записи файла INVENTORY-FILE и переписывать их в записи файла ABSTRACT-FILE, имеющие следующее описание:

```
01 ABSTRACT-RECORD.
  05 ABSTRACT-ITEM-NUMBER        PICTURE IS X(8).
  05 ABSTRACT-NUMBER-ON-HAND     PICTURE IS X(6).
```

Пример В

Предположим, что колода состоит из перфокарт, на которых в колонках с 1 по 35 пробито полное имя сотрудника, а в колонках с 36 по 44 пробит номер его карточки социального обеспечения (номер из девяти цифр). Напишите полную КОБОЛ-программу для считывания этих карт и распечатывания их с определенными изменениями в виде списка. Вставьте в номер карточки социального обеспечения дефисы между третьей и четвертой, а также между пятой и шестой цифрами. Кроме того, в строке печати расположите номер карточки социального обеспечения первым и отделите его от имени 10 пробелами, например

826-94-5688 JOHN R. SMITH

Пример С

Предположим, что файл хранится на катушке магнитной ленты и является набором записей, содержащих некоторое количество анкетной информации, организованной следующим образом:

Позиции литер	Описание данного	Пример значения
1—25	фамилия	SMITH
26—35	имя	JOHN
36	инициал	R
37—56	адрес	351 D AVENUE
57—71	город	AMES
72—73	штат	PA
74—78	почтовый индекс	19104
79—82	код отдела	3385
83—88	годовой оклад	015500

Напишите полную КОБОЛ-программу для составления отчета о личном составе, названного LIST OF PERSONNEL (СПИСОК СОТРУДНИКОВ), путем распечатывания всех записей файла. В отчете должны быть представлены только PERSONNEL NAME (ИМЯ СОТРУДНИКА), DEPARTMENT CODE (КОД ОТДЕЛА) и ANNUAL SALARY (ГОДОВОЙ ОКЛАД).

Решение А

PROCEDURE DIVISION.

START-X.

OPEN INPUT INVENTORY-FILE OUTPUT
ABSTRACT-FILE.

LOOP-X.

READ INVENTORY-FILE RECORD AT END
CLOSE INVENTORY-FILE ABSTRACT-FILE
STOP RUN.

MOVE ITEM-NUMBER TO ABSTRACT-ITEM-NUMBER.
MOVE NUMBER-ON-HAND TO
ABSTRACT-NUMBER-ON-HAND.

WRITE ABSTRACT-RECORD.
GO TO LOOP-X.

Решение В

IDENTIFICATION DIVISION.

PROGRAM-ID. B55.

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER. MARK-1.

OBJECT-COMPUTER. MARK-1.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

SELECT CARD-IN ASSIGN TO CARD-READER.

SELECT PRINT-OUT ASSIGN TO PRINTER.

DATA DIVISION.

FILE SECTION.

FD CARD-IN LABEL RECORD IS OMITTED.

01 CARD-IN-RECORD.

05 CARD-NAME PICTURE IS X(35).

05 CARD-SOCIAL-SECURITY-NUMBER.

10 DIGITS-ONE-TO-THREE PICTURE IS X(3).

10 DIGITS-FOUR-AND-FIVE PICTURE IS X(2).

10 DIGITS-SIX-TO-NINE PICTURE IS X(4).

05 FILLER PICTURE IS X(36).

FD PRINT-OUT LABEL RECORD IS OMITTED.

01 PRINT-OUT-RECORD.

05 PRINT-SOCIAL-SECURITY-NUMBER.

10 PRINT-ONE PICTURE IS X(3).

10 PRINT-TWO PICTURE IS X.

10 PRINT-THREE PICTURE IS X(3).

10 PRINT-FOUR PICTURE IS X.

10 PRINT-FIVE PICTURE IS X(4).

05 SPACE-X PICTURE IS X(10).

05 PRINT-NAME PICTURE IS X(35).

PROCEDURE DIVISION.

START-X.

OPEN INPUT CARD-IN OUTPUT PRINT-OUT.

LOOP-X.

READ CARD-IN RECORD AT END CLOSE CARD-IN
PRINT-OUT STOP RUN.

MOVE DIGITS-ONE-TO-THREE TO PRINT-ONE.

MOVE "-" TO PRINT-TWO.

MOVE DIGITS-FOUR-AND-FIVE TO PRINT-THREE.

MOVE "-" TO PRINT-FOUR.

MOVE DIGITS-SIX-TO-NINE TO PRINT-FIVE.

MOVE SPACES TO SPACE-X.

MOVE CARD-NAME TO PRINT-NAME.

WRITE PRINT-OUT-RECORD.
GO TO LOOP-X.

(Заметим, что использование оператора

MOVE "-" TO PRINT-TWO PRINT-FOUR.

позволило бы сэкономить одну строку в исходной программе.)

Решение С

IDENTIFICATION DIVISION.

PROGRAM-ID. C55.

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER. MARK-1.

OBJECT-COMPUTER. MARK-1.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

SELECT PERSONNEL-FILE ASSIGN TO MAGNETIC-TAPE.

SELECT PRINT-FILE ASSIGN TO PRINTER.

DATA DIVISION.

FILE SECTION.

FD PERSONNEL-FILE LABEL RECORDS ARE STANDARD.

01 PERSONNEL-RECORD.

05 PERSONNEL-NAME PICTURE IS X(36).

05 FILLER PICTURE IS X(42).

05 PERSONNEL-DEPARTMENT PICTURE IS X(4).

05 PERSONNEL-SALARY PICTURE IS X(6).

FD PRINT-FILE LABEL RECORDS ARE STANDARD.

01 PRINT-RECORD.

05 PRINT-NAME PICTURE IS X(36).

05 FILLER PICTURE IS X(5).

05 PRINT-DEPARTMENT PICTURE IS X(4).

05 FILLER PICTURE IS X(5).

05 PRINT-SALARY PICTURE IS X(6).

WORKING-STORAGE SECTION.

01 NAME-OF-REPORT.

05 FILLER VALUE IS SPACES PICTURE IS X(20).

05 FILLER PICTURE IS X(19)

VALUE IS "LIST OF PERSONNEL".

01 HEADER-OF-REPORT.

05 FILLER PICTURE IS X(14)
VALUE IS "PERSONNEL NAME".
05 FILLER VALUE IS SPACE PICTURE IS X(16).
05 FILLER PICTURE IS X(15)
VALUE IS "DEPARTMENT CODE".
05 FILLER VALUE IS SPACE PICTURE IS X(5).
05 FILLER PICTURE IS X(13)
VALUE IS "ANNUAL SALARY".

PROCEDURE DIVISION.

START-X.

OPEN INPUT PERSONNEL-FILE OUTPUT
PRINTER-FILE.

WRITE PRINT-RECORD FROM NAME-OF-REPORT.

WRITE PRINT-RECORD FROM HEADER-OF-REPORT.

MOVE ALL SPACES TO PRINT-RECORD.

LOOP-X.

MOVE ALL SPACES TO PRINT-RECORD.

WRITE PRINT-RECORD.

READ PERSONNEL-FILE RECORD AT END CLOSE
PERSONNEL-FILE PRINTER-FILE
STOP RUN.

MOVE PERSONNEL-NAME TO PRINT-NAME.

MOVE PERSONNEL-DEPARTMENT TO PRINT-
DEPARTMENT.

MOVE PERSONNEL-SALARY TO PRINT-SALARY.

GO TO LOOP-X.

3.6. Редактирование с помощью перемещения

Возможность при элементарном перемещении осуществлять любое необходимое преобразование одной формы внутреннего представления данных в другую является очень полезной, хотя иногда и приводит в замешательство. Все групповые перемещения рассматриваются как нечисловые и для них не производится никакого редактирования, хотя усечение или заполнение пробелами можно было бы рассматривать как форму простого редактирования. Другую форму простого редактирования представляет собой выравнивание относительно подразумеваемой десятичной точки и любое дополнение нулями, необходимое для окончательного формирования

принимающего данного при числовом перемещении. Более сложное редактирование заключается в подготовке данного для хорошего наглядного представления в печатной форме, как, например, вставление явной десятичной точки, отбрасывание ведущих нулей или внесение валютного знака. Все эти действия регулируются фразой PICTURE принимающего данного, лишь две формы которой были упомянуты до сих пор, а именно: было сказано, что строка-литер этой фразы состоит из символов X для нечисловых данных и из символов 9 и, возможно, символов S (3), V (T) и P (M) для числовых данных.

Групповые перемещения

Групповое перемещение имеет место, когда либо пересылаемое, либо принимающее данное является групповым. При групповом перемещении литеры просто переносятся одна за другой, начиная слева, и пересылаемое данное либо усекается, либо дополняется справа пробелами в зависимости от соотношения длин данных. При групповом перемещении фразы PICTURE пересылаемого и принимающего данного используются только для определения их длин. То, что какая-то из фраз PICTURE может определять числовое данное, не влияет на групповое перемещение. Пусть определено групповое данное:

01 X-REC.

05 A-ITEM PICTURE IS 9(3)V9(2).

05 B-ITEM PICTURE IS 9(3)P(3).

05 C-ITEM PICTURE IS S9(5)V99.

Общая длина группового данного X-REC равна пятнадцати литерным позициям, при этом длина A-ITEM составляет пять позиций, длина B-ITEM — три позиции и длина C-ITEM — семь позиций. Для символов V, P и S в поле данного не выделяется никаких литерных позиций. Значением данного X-REC могло бы быть, например, такое

032501990320065⁺

которое представляло бы значения 32.50 для A-ITEM, 199000 для B-ITEM и +3200.65 для C-ITEM. Если некоторый оператор перемещает данное X-REC в другое данное, имеющее по крайней мере пятнадцать литерных позиций, то будут переданы все пятнадцать литер данного X-REC. Это групповое перемещение может привести к изменению ожидаемых значений принимающих данных, если строки-литер фраз PICTURE не совпадают полностью. Например, если бы принимающее данное было описано так:

01 Y-REC.

05 D-ITEM PICTURE IS 9(4)V9.

05 E-ITEM PICTURE IS 9(3)P(3).

05 F-ITEM PICTURE IS S9(3)V9(3).

то оператор

MOVE X-REC TO Y-REC

поместил бы пятнадцать литер из X-REC в Y-REC. Литеры данного D-ITEM были бы таковы: 03250, но его значение равнялось бы 325.0, так как при групповом перемещении не происходит выравнивания относительно десятичной точки. Оператор

MOVE A-ITEM TO D-ITEM

задает элементарное перемещение, в результате которого данное D-ITEM оказалось бы состоящим из литер 00325 и имело бы значение 32.5, совпадающее со значением данного A-ITEM.

Категории редактирования

Элементарное перемещение имеет место, когда оба данных элементарные. Если оба поля, и пересылаемое, и принимающее, описаны фразами PICTURE как числовые, то при перемещении происходит выравнивание относительно подразумеваемой десятичной точки и дополнение нулями. Это простая форма редактирования. Такое редактирование выполняется только при элементарном перемещении. Редактирование регулируется фразой PICTURE принимающего данного.

Существуют различные типы редактирования. Простейший тип редактирования заключается в дополнении пробелами: если размер принимающего данного больше размера пересылаемого данного, то лишние позиции заполняются пробелами. Следующий тип — это выравнивание с дополнением нулями, упомянутое в предыдущем абзаце. Затем следует простая вставка, при которой пробел или ноль могут быть автоматически вставлены внутрь последовательности литер. Этот последний тип редактирования допускает вставку различных литер, подавление ведущих нулей и другие действия, которые будут описаны ниже в этом разделе.

Различные уровни редактирования требуют более детальной классификации принимающих данных, чем простое деление на нечисловые и числовые данные. Ниже приводится дальнейшая классификация данных.

Значения данных

1. числовое

а) числовое целое

б) числовое нецелое

2. нечисловое

- а) буквенное
- б) буквенно-цифровое
- в) буквенно-цифровое редактируемое
- г) числовое редактируемое

Категория данного определяется фразой PICTURE, записанной в разделе DATA DIVISION. Фраза PICTURE может быть использована только на элементарном уровне. Она должна указываться в статье описания элементарного данного. Эта фраза состоит из ключевого слова PICTURE, за которым следует строка-литер. Максимальное число литер в этой строке равно тридцати, но это не ограничивает размер описываемого данного, так как можно задавать число повторений литеры, заключенное в круглые скобки, например

PICTURE IS 9(5)V9(2)

означает то же самое, что и

PICTURE IS 99999V99

Общий формат фразы PICTURE таков:

$$\left\{ \begin{array}{c} \text{PICTURE} \\ \hline \text{PIC} \end{array} \right\} \text{ IS строка-литер}$$

где PIC (Ш) — это возможное сокращение слова PICTURE (ШАБЛОН). Внутри строки-литер не могут появляться пробелы.

Числовые категории

Строка-литер фразы PICTURE, определяющая числовое данное, может содержать только символы 9, P (М), S (З) и V (Т). Если присутствует символ S, то он должен быть крайним слева символом в строке-литер фразы PICTURE. Он указывает на наличие знака числа. Символ S не учитывается при определении размера данного, а сам знак будет совмещен с крайней правой цифрой данного. Каждый символ 9 в строке шаблона представляет позицию литеры в данном, которая содержит цифру, и является единственным символом в строке шаблона, который учитывается при определении размера данного. Символ P означает масштабирование для определения положения подразумеваемой десятичной точки. Символы P могут находиться как слева, так и справа от символов 9, но не могут быть расположены одновременно с двух сторон, так как в числе десятичная точка только одна. Символ V указывает на положение, подразумеваемой десятичной точки и может появляться только один раз. Он может быть опущен, когда данное описывается как целое число или когда присутствуют символы P, так как в этих случаях положе-

ние десятичной точки определяется автоматически. Отметим снова, что в самом данном нет десятичной точки; данное может содержать только десятичные цифры и знак числа. Примеры числовых данных:

Литеры, хранящиеся во внутренней памяти	Фраза PICTURE	Значение данного
12345	9(5)V	12345.
12345	9(5)	12345.
12345	9(4)V9	1234.5
12345	PP9(5)	.0012345

Числовые данные можно далее разбить на две категории: числовые целые и числовые нецелые. Числовое целое — это данное, в котором подразумеваемая десятичная точка находится непосредственно справа от крайней правой хранящейся цифры, т. е. это целое число, полностью представленное хранящимися в памяти литерами. Во фразе PICTURE для такого данного может содержаться или нет символ знака, символ V может быть опущен как лишний, а символы P не должны присутствовать. Все следующие примеры являются примерами числовых целых:

Литеры, хранящиеся во внутренней памяти	Фраза PICTURE	Значение данного
$\overline{347}$	S999V	—347.
347	999V	347.
$\overset{+}{347}$	S999	+347.
$\overline{347}$	S999	—347.

Числовое нецелое — это более общая категория, в которой подразумеваемая десятичная точка может находиться в любом месте, как было описано ранее для числовых данных, например:

346 $\overline{5}$	S9(3)V9	—346.5
346	999PPP	346000.

Обратите внимание, что данное в последнем примере не является числовым целым, несмотря на то что его значение — целое число. В числовом целом не допускается масштабирование.

Дополнение нулями также является одним из типов редактирования, которое может применяться к принимающим данным, относящимся к категории числовых целых данных. Когда в такое данное помещается число, оно будет дополнено ведущими нулями, чтобы целиком заполнить данное десятичными цифрами. Следовательно, если бы значение 35.0 нужно было поместить в данное, например, определенное как

01 NUMERIC-INTEGGER-ITEM PICTURE IS 99999.

то фактически в данное NUMERIC-INTEGGER-ITEM были бы записаны цифры 00035.

К принимающим данным, относящимся к категории числовых нецелых, могут быть применены десятичное выравнивание и дополнение нулями. Когда число помещается в такое данное, его значение будет размещено в пределах выделенного поля в соответствии с положением подразумеваемой десятичной точки и будет добавлено необходимое число нулей. Следовательно, если бы значение 68.5 нужно было поместить в данное, определенное как

01 NUMERIC-NON-INTEGGER-ITEM PICTURE IS 9999V99.

то фактически в данное NUMERIC-NON-INTEGGER-ITEM были бы записаны цифры 006850.

Буквенная категория

Категория данных, являющихся нечисловыми, значения которых составляются из сорока шести букв русского и латинского алфавитов и пробела, называется буквенной категорией. Примером такого данного является имя JOHN SMITH. Фраза PICTURE для буквенных данных может содержать только символы А и В. Это новые символы, не упоминавшиеся до сих пор. Каждый из символов А и В в строке-литер фразы PICTURE учитывается при определении размера данного. Символ А означает, что соответствующая позиция литеры в данном может содержать только букву или пробел. Например, данное

01 ALPHABETIC-CATEGORY-ITEM PICTURE IS A(10).

может иметь такие значения:

ABCDEFGHIJ
JOHN SMITH
A B C D E

Значением, не допустимым для этой категории, было бы \$12345.67#*.

Символ В во фразе PICTURE — это символ редактирования, который приводит к вставке пробела в соответствующую позицию

литеры принимающего данного. Это именно вставка, а не замещение, и размер принимающего данного должен быть больше размера пересылаемого данного на число символов В; в противном случае значение будет усечено справа. Например, если бы выполнялся оператор

MOVE "ABCDE" TO ITEM-B-X

то для различных фраз PICTURE данного ITEM-B-X были бы получены следующие результаты:

Данное ITEM-B-X	
Хранившиеся литеры	Фраза PICTURE
AB CDE	A(2)BA(3)
A B C D E	ABABABABA
AB CDE	A(2)BABA(2)

Буквенная категория нечисловых данных используется только в том случае, когда программист уверен, что в описанных таким образом данных будут появляться только буквы и пробелы. Например, адрес места жительства не будет относиться к буквенной категории, так как он включает номера дома и квартиры.

Буквенно-цифровая категория

Другая категория нечисловых данных — это буквенно-цифровая категория. Буквенно-цифровые данные могут состоять из любых литер, допустимых на данной рабочей машине. Их число может достигать до 256 литер, включая заглавные и строчные буквы. Фраза PICTURE, идентифицирующая эту категорию, обычно состоит из символов X, но может включать и символы A и 9. Примеры буквенно-цифровых значений таковы:

THEY SAID! @#%#&#&
John R. Smith,
125 N. BOLTON ST.

Единственное редактирование, имеющее место в случае, когда принимающее данное буквенно-цифровое и длиннее, чем пересылаемое данное, состоит в дополнении пробелами. Для данного, описанного с помощью статьи

01 ALPHANUMERIC-DATA-ITEM PICTURE IS X(6).

оператор

MOVE "ABC" TO ALPHANUMERIC-DATA-ITEM

занес бы в это данное три буквы и три пробела: ABC . Заметьте, что в примерах буквенных и буквенно-цифровых данных

кавычки в операторах MOVE не являются частью литерала и не помещаются в принимающее данное. Если пересылаемое данное слишком длинно для того, чтобы разместиться в принимающем поле, то лишние литеры усекаются справа. Поэтому в результате выполнения оператора

MOVE "ABCDEFGH" TO ALPHANUMERIC-DATA-ITEM

значение принимающего данного оказалось бы равно ABCDEF. Для буквенно-цифровой категории принимающих данных не применяется никакого другого редактирования.

Строка-литер шаблона для буквенно-цифровой категории наряду с символами X может содержать символы A и 9, но данное обрабатывается так, как если бы все эти символы были символами X. Следовательно, шаблон

PICTURE IS A(5)9(2)X(2)

обрабатывается точно так же, как если бы он был записан в виде

PICTURE IS X(9).

Единственная причина для использования символов A и 9 заключается в том, чтобы информировать программиста об определенной структуре данного, например

05 SOCIAL-SECURITY-NUMBER PICTURE IS 9(3)X99X9(4)
VALUE IS "123-45-6789".

Если бы вся строка литер шаблона состояла из одних символов 9 или из одних символов A, то данное относилось бы к категории числовых целых или к категории буквенных данных, а не к категории буквенно-цифровых данных. Начинающему программисту рекомендуется использовать только символ X для всех данных буквенно-цифровой категории.

Буквенно-цифровая редактируемая категория

Существует другая категория нечисловых данных, которая допускает некоторое редактирование буквенно-цифровых значений. Управляющими символами в строке-литер шаблона могут быть любые из следующих: A, X, B, 9 или 0 (цифра ноль). Для этой категории редактирование ограничено простой вставкой, как и для буквенной категории, но данные могут содержать любые литеры, а не только буквы, и вставлять можно как пробелы, так и нули. Для рассматриваемой категории допускаются только определенные комбинации управляющих символов. Строка-литер шаблона должна содержать:

- 1) по крайней мере один символ В и по крайней мере один символ X, или
- 2) по крайней мере один символ 0 и по крайней мере один символ X, или
- 3) по крайней мере один символ 0 и по крайней мере один символ A.

Возможные строки-литер фразы PICTURE для буквенно-цифровых редактируемых данных:

PICTURE IS X(5)BBVXBVBX(5).

PICTURE IS 99BBA(5)BV000.

Ниже приводятся примеры редактирования вставкой во время перемещения:

Пересылаемое данные	Принимающая фраза PICTURE	Принимающее данные
JAMES SMITH A35 123456789	X(5)BX(5) X(3)0(3) X(2)BX(2)BX(5)	JAMES SMITH A35000 12 34 56789

Литера пробела вставляется в позицию литеры, соответствующую символу В, а литера нуля вставляется в позицию, соответствующую символу 0. Все литеры являются буквенно-цифровыми. В последнем примере результат 12 34 56789 не является числовым и не может быть использован в арифметических операциях.

Упражнения

1. Для приведенных ниже статей-описания-записей и следующих за ними операторов MOVE запишите в свободных клеточках литерные значения, которые получаются в результате выполнения перечисленных операторов:

01 RECORD-HOLDING-VARIOUS-ITEMS.

05 NUMERIC-INTEGERS

PICTURE IS 9(5).

05 NUMERIC-NON-INTEGERS

PICTURE IS 9(3)V99.

05 ALPHABETIC-ITEM

PICTURE IS
B(3)A(10)B(3).

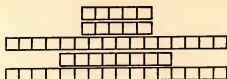
05 ALPHANUMERIC-ITEM

PICTURE IS X(8).

05 ALPHANUMERIC-EDITED-ITEM

PICTURE IS
X(2)BX(2)BX(10).

Распределение места в памяти



NUMERIC-INTEGER
 NUMERIC-NON-INTEGER
 ALPHABETIC-ITEM
 ALPHANUMERIC-ITEM
 ALPHANUMERIC-EDITED-ITEM

MOVE 65 TO NUMERIC-INTEGER.
 MOVE 38.6 TO NUMERIC-NON-INTEGER.
 MOVE "JOHN SMITH" TO ALPHABETIC-ITEM.
 MOVE "SM-4CL" TO ALPHANUMERIC-ITEM.
 MOVE "J.R.SMITH" TO ALPHANUMERIC-EDITED-ITEM.

2. Предположим, что в разделе DATA DIVISION задана следующая статья-описания-записи:

01 JOB-FILE-RECORD.

05 NAME-X

PICTURE IS A(17).

05 ADDRESS-X

PICTURE IS X(30).

05 SOCIAL-SECURITY-X

PICTURE IS 9(9).

05 SALARY-X

PICTURE IS 9(5)V99.

Данное NAME-X состоит из первых двух инициалов имени человека, за которыми непосредственно следует до пятнадцати символов фамилии, например JRSMITH. Напишите статью-описания-записи для выходной записи, которая будет содержать ту же информацию, что и запись JOB-FILE-RECORD, но перестроенную следующим образом:

- 1) должен быть пробел между первым и вторым инициалами и между вторым инициалом и фамилией;
- 2) адрес не меняется;
- 3) в номер карточки социального обеспечения должен быть вставлен пробел после второй и после пятой цифры, например: 12 345 6789;
- 4) оклад должен быть масштабирован таким образом, чтобы устранить пенни.

3.7. Основной процесс редактирования: числовая редактируемая категория

Пять категорий данных, описанные до сих пор, допускают сравнительно незначительное редактирование при выполнении оператора MOVE. Шестая, и последняя, категория допускает несколько типов редактирования, обеспечивающих преобразование числовых значений в печатную форму, удобную для легкого чтения и быст-

рого понимания. Категория называется числовой редактируемой, но важно помнить, что она относится к нечисловому типу. Числовые редактируемые данные нельзя использовать ни в каких арифметических операциях. Такие данные определяются строкой-литер шаблона, содержащей определенные комбинации следующих символов:

B P V Z 0 9 , . * + — \$ CR DB
(B M T П 0 9 , . * + — □ KP DB)

Этот список содержит наряду со знакомыми несколько новых символов. Отсутствуют в этом списке и не могут быть использованы в строке-литер шаблона для описания числовых редактируемых данных символы A, S и X. Эти три символа относятся к буквенным литерам, знаку числа и буквенно-цифровому набору литер. Назначение числовой редактируемой категории заключается в том, чтобы представить числовую информацию в формате, ориентированном на коммерческие приложения, путем включения явных десятичных точек, запятых, валютных символов и других символов, используемых в отчетах. Например, примерами числовых редактируемых данных будут:

\$31,650.50CR
\$***850.00
4,500

В памяти будут фактически храниться те литеры, которые показаны в примерах; первой литерой будет валютный знак, следующими двумя литерами будут цифры 3 и 1, за которыми следует явная запятая и т. д. Числовая редактируемая категория предназначена для представления данных при выводе их на печатающее устройство.

Наличие управляющих символов во фразе PICTURE для некоторого данного определяет формат этого данного после занесения в него значения. Для каждого из символов определены смысл и ограничения на его использование. Не все символы могут быть использованы в одной строке-литер шаблона. Некоторые из символов могут появляться только один раз, это относится к символам: V, ., CR и DB. Смысл символов B, P, V, 0 и 9 уже был объяснен и остается прежним. Строка-литер должна описывать данное, которое будет содержать цифры и определенные описательные литеры, облегчающие чтение числа. Для того чтобы данное было классифицировано как числовое редактируемое, строка-литер его шаблона должна содержать по крайней мере один из следующих символов:

B Z 0 * + — , . \$ CR DB

Здесь пары литер CR и DB рассматриваются как один символ. В каждой строке-литер должен встречаться по крайней мере один символ 9, а общее число этих символов (с учетом повторений) не должно превышать восемнадцать.

Редактирование вставкой

Вставление некоторой литеры в принимающее данное называется редактированием вставкой. В основном существует два возможных типа вставок: фиксированная вставка и плавающая вставка. Результатом редактирования фиксированной вставкой является занесение литеры в заданную позицию данного. Результатом редактирования плавающей вставкой является занесение литеры непосредственно перед первой ненулевой цифрой, считая слева.

Каждый символ шаблона В или 0 вызывает вставку пробела или нуля. Запятая действует таким же образом, приводя к вставке явной запятой в те позиции данного, которые соответствуют позициям запятых в строке-литер шаблона. Десятичная точка представляет собой символ специальной вставки, так как она не только приводит к вставке явной десятичной точки в принимающее данное, но и вызывает десятичное выравнивание, так что цифры располагаются в соответствии с их значениями. Десятичная точка, используемая совместно с символами запятой и нуля, может облегчить чтение чисел, например:

Пересылаемое данное		Принимающее данное	
PICTURE	Значение	PICTURE	Значение
9(4)V9	12345	9,999.99	1,234.50
		9,999.00	1,234.00
		9999.99	1234.50
9(4)V9	00123	9,999.99	0,012.30

Заметьте, что запятая в последнем примере вставлена между двумя нулями, так как используется при редактировании фиксированной вставкой. Символ V также можно использовать в строке-литер шаблона, но не одновременно с десятичной точкой. Символ V не приводит к вставке, но вызывает расположение цифр в соответствии с их значениями.

Четыре символа +, —, CR и DB — это символы, управляемые знаком; они представляют позиции литер, в которых будет размещен знак числа. Напомним, что символ S в числовой редактируемой категории не допускается, и другие символы в известном смысле заменяют его. Символы CR и DB задают способ изображения знака числа в виде кредита или дебета, которые удобны для бухгалтерских задач. Символы + и — задают более математические обозначения для положительных и отрицательных чисел. Эти четыре символа взаимно исключают друг друга. В каждой строке-литер шаблона может присутствовать только один из них. Символы кредита и дебета также могут содержаться в строке-литер только один раз и при этом должны быть самыми правыми. Каждый из символов CR и DB соответствует двум позициям литер в поле данного, описывае-

мого с помощью строки-литер фразы PICTURE. В эти две позиции литер заносятся пробелы, если данное положительно, и заносятся пары литер CR или DB (в соответствии с тем, какой символ указан в шаблоне), если данное отрицательно. Заметьте, что пробелы заносятся для положительных значений, а пара букв, та или другая, заносится для отрицательных значений данного. Ошибочно считать, что символ CR означает плюс, а символ DB означает минус. Они оба означают минус.

При использовании символов + и — допускается большая гибкость. Эти символы можно употреблять в виде одиночных символов, расположенных в самой правой или самой левой позиции, либо в виде цепочки таких символов, расположенной слева. Этот последний случай приводит к редактированию плавающей вставкой, которое будет описано в скором времени. При редактировании фиксированной вставкой в строке-литер шаблона может появляться лишь один такой символ. Различие в действии символов + и — заключается в том, что в случае, когда пересылаемое данное положительно, символ + вызывает занесение явного знака плюс, в то время как символ — вызывает занесение пробела. Если пересылаемое данное отрицательно, то оба символа вызывают занесение минуса. Оба этих символа не влияют на значение данного, они лишь управляют тем, какая литера будет вставлена в данное. Рассмотренные комбинации сведены в таблицу:

Фактический результат

Редактирующий символ в строке-литер шаблона	Данное положительно или нуль	Данное отрицательно
+	+	—
—	пробел	—
CR	2 пробела	CR
DB	2 пробела	DB

Примеры редактирования, управляемого знаком:

PICTURE	Пересылаемое данное	PICTURE	Принимающее данное	Размер
9(5)	12345	9(5)CR	12345	7
S9(5)	-12345	9(5)CR	12345CR	7
9(5)	12345	+9(5)	+12345	6
9(5)	12345	-9(5)	12345	6
9(3)V9(2)	12345	+9(3).99	+123.45	7
9(3)V9(2)	00045	+9(3).99	+000.45	7

Вновь обратите внимание, что при редактировании фиксированной вставкой в принимающем данном присутствуют нули.

Другим типом редактирования вставкой является редактирование плавающей вставкой. При этом используются управляющие символы \$(O)\$, + и —. Как правило, они никогда не используются вместе в одной строке-литер шаблона, но единственный символ, управляемый знаком, может быть использован в качестве либо самого левого, либо самого правого символа. Ниже предполагается, что эти три плавающих символа являются взаимно исключающими. При этом результаты одинаково применимы к каждому символу с той лишь разницей, что для каждого управляющего символа в принимающее данное будет вставлена своя литера. Символы + и — приводят к вставке литер плюс или минус в соответствии с упомянутыми выше правилами. Символ \$ приводит к вставке литеры \$ или какого-либо иного зарезервированного в трансляторе валютного знака, например, для задач, включающих обработку иностранных валют. Когда в строке-литер шаблона используется только один плавающий символ, литера будет вставлена в соответствующую позицию литеры в данном. Однако, когда используется более одного плавающего символа, подходящая литера будет вставлена непосредственно слева от первой ненулевой цифры данного. Также важно, что нули, запятые и десятичная точка, расположенные левее позиции плавающей вставки, все будут заменены пробелами, например:

Пересылаемое данное		Принимающее данное	
PICTURE	Значение	PICTURE	Значение
9(5)	12345	\$9(5)	\$12345
9(3)V99	00123	\$9(3).99	\$001.23
9(5)	12345	\$\$\$\$\$	\$12345
9(5)	00123	\$(6)	\$123
9(3)V99	00123	\$(4).9(2)	\$1.23
9(3)V99	00001	\$(4).\$(2)	\$.01
9(3)V99	00000	\$(4).\$(2)	
9(4)V99	123456	\$\$,\$\$\$.\$	\$1,234.56
9(4)V99	001234	\$\$,\$\$\$.\$	\$12.34
9(3)V99	12345	+(4).9(2)	+123.45
9(3)V99	00012	+(4).9(2)	+12
9(3)V99	00000	+(4).+(2)	

Когда в пересылаемом данном встречается ненулевая цифра, эта цифра сама переносится в принимающее данное, так что при просмотре данного слева направо для ненулевых цифр плавающие символы эквивалентны символу 9. Если плавающие символы встречаются

ся справа от десятичной точки, то в случае нулевого численного значения, помещаемого в принимающее данное, оно будет все заполнено пробелами.

Редактирование подавлением нулей

Валютный символ \$ и символы + и —, управляемые знаком, заменяют ведущие нули, запятые и десятичную точку пробелами. Исключение составляет лишь позиция, расположенная непосредственно слева от первой ненулевой цифры. Эта позиция заполняется самим плавающим символом.

Существует другой метод подавления ведущих нулей, не использующий эти плавающие символы. Имеются два управляющих символа Z (П) и *, которые называются символами подавления нулей. Они действуют лишь до некоторой степени подобно плавающим символам. Отличия касаются главным образом нулей, расположенных справа от явной десятичной точки. Управляющий символ Z замещает пробелами литеры, находящиеся левее первой ненулевой цифры, так что данное 00123 при размещении в принимающем поле примет вид 123. Символ подавления * замещает ведущие литеры звездочкой. Такой способ изображения, например, используется при печати суммы на чеке, когда желательно защитить ее от подделок (например, от приписывания слева значащих цифр). Данное 00123 в результате такого редактирования приняло бы вид **123. Если при просмотре данного слева направо найдена ненулевая цифра, то для нее символы подавления действуют точно так же, как символ 9. В любой строке-литер символы Z и * являются взаимно исключающими. Каждый символ подавления учитывается при определении размера данного. Такой символ может находиться не только в ведущих позициях, но и в любых позициях, включая позиции, расположенные правее десятичной точки, например:

ZZZ.99 или ZZZ.ZZ или *****.

Если символы подавления встречаются только слева от явной десятичной точки (или подразумеваемой десятичной точки V), то любое подавление нулей и запятых будет заканчиваться на этой десятичной точке или на первой ненулевой цифре в зависимости от того, что раньше встретится. Если все цифровые позиции, включая расположенные правее десятичной точки, заполнены символами подавления, то подавление прекращается на десятичной точке, если только данное не равно в точности нулю. В последнем случае значением принимающего поля будут:

1. Все пробелы, если используется Z.
2. Все *, за исключением десятичной точки, если используется *.

Примеры использования символов подавления:

Пересылаемое данное		Принимающее данное	
PICTURE	Значение	PICTURE	Значение
9(6)	001234	Z(6)	1234
9(6)	012345	Z(6)	12345
9(6)	123456	Z(6)	123456
9(3)V9(3)	123456	Z(3).Z(3)	123.456
9(3)V9(3)	001234	Z(3).Z(3)	1.234
9(3)V9(3)	000001	Z(3).Z(3)	.001
9(3)V9(3)	000000	Z(3).Z(3)	
9(3)V9(3)	000000	* (3).*(3)	***.***
9(3)V9(3)	001234	* (3).*(3)	**1.234

Перемещения, допустимые в КОБОЛе

Рассмотренные шесть категорий (буквенная, буквенно-цифровая, буквенно-цифровая редактируемая, числовая редактируемая, числовая целая и числовая нецелая) допускают не только простые перемещения числового в числовое или нечислового в нечисловое. Например, числовое целое можно поместить в числовое редактируемое, превратив его тем самым в строку литер, удобную для выдачи. При этом значение 890 могло бы стать, например, таким: \$890.00. Однако обратное преобразование невозможно, так как в КОБОЛе предусмотрена только вставка и нет удаления специальных литер. Многие комбинации перемещений не допускаются. Необходимость помнить, какие из комбинаций допустимы, а какие нет, часто вызывает у программиста головную боль. На рис. 3.4 представлена таблица допустимых в КОБОЛе перемещений. Пустые позиции в таблице означают, что соответствующее перемещение запрещено. Позиции «только цифры» и «только буквы» означают, что соответствующее перемещение допустимо только для пересылаемых данных, состоящих именно из этих литер. Трудно дать какие-либо простые правила для запоминания допустимых в КОБОЛе перемещений, но во всех компиляторах предусматриваются встроенные средства контроля для обнаружения запрещенных перемещений и предостережения программиста от попыток использовать недопустимые перемещения данных.

Ниже приводится список управляющих символов для строки-литер шаблона:

Символ	Значение
B (B)	Вставка пробела.
P (M)	Десятичная позиция, расположенная вне данного.
V (T)	Позиция подразумеваемой десятичной точки.

Z	(П)	Подавление нуля.
0	(0)	Вставка нуля.
9	(9)	Позиция цифры.
,	(,)	Вставка запятой.
.	(.)	Вставка явной десятичной точки.
*	(*)	Подавление нуля.
+	(+)	Управление обоими знаками.
-	(-)	Управление знаком минус.
\$	(O)	Валютный символ.
CR	(KP)	Двубуквенное указание отрицательности.
DB	(ДВ)	Двубуквенное указание отрицательности.

Пример

В следующей таблице приведены примеры результатов выполнения оператора MOVE, примененного к элементарным данным. Хранящееся значение пересылаемого данного и соответствующая строка-литер фразы PICTURE задают значение данного, которое будет помещено в принимающее данное.

Пересылаемое данное		Принимающее данное	
Строка-литер PICTURE	Хранящееся значение	Строка-литер PICTURE	Хранящееся значение
99V9(3)	12345	\$*,***.99	\$***12.34
9(5)	12345	\$\$\$,\$\$\$,99	\$12,345.00
9(5)	00000	\$\$\$,\$\$\$,99	\$,00
9(4)V9	12345	\$\$\$,\$\$\$,99	\$1,234.50
S9(5)V	12345	-ZZZZZ.99	-12345.00
S9(5)	12345	ZZZZZ.99-	12345.00-
S9(5)	12345	\$\$\$\$\$,99CR	\$12345.00CR
S9(5)	12345	\$\$\$\$\$,99CR	\$12345.00
99V99	1234	ZZZVZZ	\$1234
9(5)	12345	ZZZZZ	12345
9(5)	12345	ZZZ,ZZZ.99	\$12,345.00
9(5)	00123	ZZZ,ZZZ.99	\$123.00
9(5)	00000	ZZZ,ZZZ.99	\$,00
9(5)	00000	\$*,***.*	*****.

		ПРИНИМАЮЩАЯ КАТЕГОРИЯ					
		Буквенная	Буквенно-цифровая	Буквенно-цифровая редактируемая	Числовая редактируемая	Числовая целая	Числовая нецелая
ПЕРЕСЫЛАЕМАЯ КАТЕГОРИЯ	Нечисловое данных						
	Числовое данных						
	Буквенная	У	У	У			
	Буквенно-цифровая	только буквы	У	У	только цифры	только цифры	только цифры
	Буквенно-цифровая редактируемая	только буквы	У	У			
	Числовая редактируемая		У	У			
	Числовая целая		У	У	У	У	У
	Числовая нецелая				У	У	У

У означает допустимое перемещение.

Рис. 3.4. Перемещения, допустимые в КОБОЛе.

Упражнения

1. Укажите в двух правых колонках, задает ли оператор
MOVE SENDING-FIELD TO RECEIVING-FIELD
(ПОМЕСТИТЬ ПЕРЕСЫЛАЕМОЕ-ПОЛЕ
В ПРИНИМАЮЩЕЕ-ПОЛЕ)

допустимое перемещение или нет и какая из шести возможных категорий в строке-литер принимающего поля определяет данное. Используйте следующие сокращения:

Буквенное	Б
Буквенно-цифровое	БЦ
Буквенно-цифровое редактируемое	БР
Числовое редактируемое	ЧР
Числовое целое	ЧЦ
Числовое нецелое	ЧН

Обратите внимание, что фраза PICTURE принимающего данного определяет категорию данного независимо от того, является ли оператор MOVE допустимым или нет.

Пересылаемое поле		Принимающее поле		
Хранимое значение	Строка-литер PICTURE	Строка-литер PICTURE	Категория принимающей фразы	Допустим ли оператор или нет
ABCDE	X(5)	X(2)		
		A(5)		
		9(4)		
12345	X(5)	X(5)		
		A(3)		
		9(4)		
ABC	A(3)	A(5)		
		X(3)		
		A(2)BA		
		X(2)B9		
		9(3)		
\$1,234.00	\$\$,\$\$9.99	A(7)		
		\$\$\$,\$\$.		
		X(9)		
		BBX(6)00		
		9(4)V9(2)		
12345	9(5)	X(5)		
		X(5)000		
		\$ZZ,ZZZ		
		9(3)V99		
12345	9(3)V99	X(5)		
		BX(3)00		
		+(5).99		

2. Имеется колода карт со следующим описанием записи:

01 CARD-RECORD.

05 JOB-NUMBER.

10 SHOP-NUMBER PICTURE IS X(3).

10 SEQUENCE-NUMBER PICTURE IS X(4).

05 JOB-DATE.

10 ALPHABETIC-MONTH PICTURE IS X(3).

10 DAY-OF-THE-MONTH PICTURE IS X(2).

10 YEAR-X PICTURE IS X(4).

05 WORKER-SOCIAL-SECURITY PICTURE IS X(9).

05 TIME-JOB-STARTED.

10 STARTED-MINUTE PICTURE IS X(4).

10 STARTED-FRACTION PICTURE IS X(2).

05 TIME-JOB-FINISHED.

10 FINISHED-MINUTE PICTURE IS X(4).

10 FINISHED-FRACTION PICTURE IS X(2).

05 COST-PER-HOUP PICTURE IS X(6).

05 FILLER PICTURE IS X(27).

Каждое поле описано как буквенно-цифровое, на что указывает использование во фразах PICTURE управляющего символа X. Однако ожидается, что большинство литер будет цифрами: на самом деле только данное ALPHABETIC-MONTH будет составлено из букв. Типичная запись CARD-RECORD выглядела бы так:

5550001JUL051974018185688123255124637002550

при этом значении данных оказались бы следующими:

JOB-NUMBER	5550001
JOB-DATE	JUL051974
WORKER-SOCIAL-SECURITY	018185688
TIME-JOB-STARTED	123255
TIME-JOB-FINISHED	124637
COST-PER-HOUR	002550

В качестве упражнения напишите полную КОБОЛ-программу для занесения образов карт в файл NEW-RECORD-FILE со следующими изменениями:

а. Поместите данное JOB-NUMBER (НОМЕР-ЗАДАНИЯ) в поле из семи позиций с пробелом, разделяющим данные SHOP-NUMBER (НОМЕР-ЦЕХА) и SEQUENCE-NUMBER (НОМЕР-СЛЕДОВАНИЯ).

- б. Поместите данное JOB-DATE (ДАТА-ЗАДАНИЯ) в данное для распечатки в виде, более удобном для чтения за счет наличия пробелов.
- в. Номер социального обеспечения (WORKER-SOCIAL-SECURITY) часто записывается в виде групп цифр, разделенных пробелами. Переместите номер 018185688 так, чтобы он принял вид 018 18 5688.
- г. Преобразуйте времена начала и завершения задания (TIME-JOB-STARTED и TIME-JOB-FINISHED) к виду с явной десятичной точкой и сотыми долями минуты.
- д. Данное COST-PER-HOUR (СТОИМОСТЬ-ЧАСА-РАБОТЫ) следует представить в виде с фиксированным знаком доллара, без ведущих нулей и с вставленными запятой и точкой, например \$1,350.50.

3. Напишите полную КОБОЛ-программу для решения следующей задачи.

Имеется файл с последовательным доступом, записанный на катушке магнитной ленты и состоящий из записей подлежащих оплате счетов, содержащих следующие данные:

- а) имя поставщика из пятнадцати литер;
- б) адрес поставщика, подразделенный на части:
адрес улицы из двенадцати литер;
название города из двенадцати литер;
сокращенное название штата из двух литер;
почтовый индекс из пяти цифр.
- в) дата выписки счета, дата предоставления скидки и дата выплаты долга, каждая из которых подразделена на части:
год из четырех цифр;
месяц из двух цифр;
день из двух цифр;
- г) сумма счета, состоящая из шести цифр, представляющих суммы от одного пенса до девяти тысяч девятисот девяноста девяти долларов и девяноста девяти центов;
- д) доля скидки, выраженная в виде десятичной дроби с использованием только двух цифр для задания диапазона от 0.1 процента до 9.9 процента.

Подготовьте выходной файл с записями, сформированными для печати, в котором первые две записи — это записи заголовка, а остальные записи представляют информацию, выделенную из каждой записи подлежащего оплате счета. Выдача должна иметь такой вид:

VENDOR	ZIP	INVOICE DATE
NAME	CODE	MO DA YR
XXXXXXXXXXXXXXXXXX	XXXXX	XX XX XX

(Здесь VENDOR NAME — ИМЯ ПОСТАВЩИКА,

ZIP CODE—ПОЧТОВЫЙ ИНДЕКС
INVOICE DATE—ДАТА ВЫПИСКИ СЧЕТА.)

Для облегчения чтения данные расположены с промежутками. Обратите внимание, что порядок следования года, месяца и дня в дате выписки счета изменен и что следует печатать только последние две цифры года. Оставьте чистые строки между записями заголовка и первой строкой данных.

Глава 4. Арифметические и логические операции

4.1. Операторы ADD и SUBTRACT

Арифметические операции являются основными в процессе обработки данных так же, как операции передачи данных (с помощью операторов READ, MOVE и WRITE) и операции управления последовательностью операторов (с помощью оператора IF (ЕСЛИ)). Язык КОБОЛ обеспечивает возможность выполнения арифметических операций двумя путями: с помощью использования оператора COMPUTE (ВЫЧИСЛИТЬ) (он будет описан в следующем разделе) и с помощью операторов, использующих глаголы ADD (СЛОЖИТЬ), SUBTRACT (ОТНЯТЬ), MULTIPLY (УМНОЖИТЬ) и DIVIDE (РАЗДЕЛИТЬ). Как и для всех операторов КОБОЛа, эти глаголы должны быть первыми словами каждого из арифметических операторов. Как и ранее, операторы объединяются в предложения, заканчивающиеся точкой и одним или более пробелами, а предложения объединяются в поименованные параграфы. Параграфы могут либо объединяться, либо не объединяться в секции, но если хотя бы один параграф находится в секции, то и все параграфы должны быть организованы в секции. Арифметические операторы могут содержать литералы и имена-данных, например:

ADD 205 TO INTERMEDIATE-SUM.

MULTIPLY A-ITEM BY .0092 GIVING B-ITEM.

COMPUTE RESULTS-X = 1.05 * X-ITEM + Y-ITEM.

Литералами являются 205, .0092 и 1.05. Имена-данных — это INTERMEDIATE-SUM (ПРОМЕЖУТОЧНАЯ-СУММА), A-ITEM (ДАННОЕ-A), B-ITEM (ДАННОЕ-B), RESULTS-X (РЕЗУЛЬТАТЫ-X), X-ITEM (ДАННОЕ-X) и Y-ITEM (ДАННОЕ-Y). Литералы должны быть числовыми литералами: либо числовыми целыми, либо числовыми нецелыми. Имена-данных относятся к данным, определенным с помощью фраз PICTURE либо в секции файлов, либо в секции рабочей-памяти.

Файловая область содержит временные записи, так как оператор WRITE разрушает относящуюся к нему запись, а оператор READ записывает новую запись, затирая старую. Следовательно, более безопасно, хотя и не обязательно, выполнять арифметические операции в области рабочей-памяти. Поэтому наиболее правильным путем обработки данных является путь, состоящий в считывании информации в файловую область, перемещении ее в область рабочей-памяти, где могут быть выполнены вычисления, а затем в помеще-

нии результата в файловую область для вывода. Имена-данных, так же как и литералы, должны в основном относиться к элементарным числовым данным. Исключения будут упоминаться тогда, когда они встретятся.

Основные варианты

Допускаются определенные варианты для всех четырех простых арифметических операторов. К числу этих вариантов относятся вариант **ROUNDED** (ОКРУГЛЯЯ), вариант **GIVING** (ПОЛУЧАЯ) и вариант **ON SIZE ERROR** (ПРИ ПЕРЕПОЛНЕНИИ). Вариант **ROUNDED** включает ключевое слово **ROUNDED**, расположенное непосредственно после имени-данного, являющегося результатом арифметической операции, например:

ADD 30 TO A-ITEM **ROUNDED**;

MULTIPLY 3 BY B-ITEM **ROUNDED**;

COMPUTE C-ITEM **ROUNDED** = A-ITEM + B-ITEM.

Этот вариант не влияет на значения, участвующие в операции, а влияет только на результат. Когда результат вычислен и помещается в принимающее данное, его значение будет выровнено в соответствии с десятичной точкой, определенной во фразе **PICTURE** принимающего данного. Если в арифметическом результате число цифр справа от десятичной точки больше, чем предусмотрено фразой **PICTURE**, то лишние цифры будут отброшены, или усечены, если не используется вариант с округлением, например:

3.7, умноженное на 2.4, дает 8.88

Если шаблон принимающего данного был бы 99V9 и если бы не использовался вариант **ROUNDED**, то в качестве ответа было бы запомнено значение

088

а в случае рассматриваемого варианта ответом будет значение

089

Заметьте, что ни 088, ни 089 не является точным результатом, который равен 0888. Округление не является необходимым во всех случаях, и наличие варианта **ROUNDED** не избавляет программиста от анализа масштаба результатов и от выделения достаточного числа цифр для сохранения нужной точности. Один из возможных методов состоит в выделении под промежуточные результаты гораздо большего места, чем необходимо, и округлении лишь окончательного результата до нужного числа дробных десятичных знаков. Максимальный размер числовых данных, допустимый в КОБОЛе, составляет восемнадцать цифр.

Второй путь заключается в использовании варианта **GIVING**.

Слово GIVING эквивалентно фразе «запомнить результат в». Непосредственно за ключевым словом GIVING следует имя-данного, порождая оператор вида:

ADD A B GIVING C

Результаты будут запоминаться в C, а значения остальных данных изменяться не будут. Особенность варианта GIVING заключается в том, что определяемое им принимающее данное не обязательно

Статья-описание-данных:

01 EXAMPLE-SET-OF-ITEMS.

05 A	PICTURE IS 9V99	VALUE IS 1.0.
05 B	PICTURE IS S99V9	VALUE IS -.5.
05 C	PICTURE IS S999V99	VALUE IS +600.5.
05 D	PICTURE IS ++,+++.99.	

Литеры, первоначально хранящиеся в данных A, B и C:

100
050
60050

В результате выполнения
этих операторов

Будут получены
следующие значения

	A	B	C	D
ADD 5.099 TO A	609			
ADD 5.099 TO A ROUNDED	610			
ADD -.5 TO A	400			
ADD 5.2 TO B		002		
SUBTRACT -5.2 FROM B		002		
ADD 200 TO B		950		
ADD A B TO C			59650	
ADD 600 C GIVING D				+1,200.50
SUBTRACT 605 FROM C GIVING D				-4.50

Рис. 4.1. Примеры операторов ADD и SUBTRACT. (Отсутствие значений в столбцах, соответствующих данным A, B, C и D, означает, что в этом случае сохраняется исходное значение данного. Во всех операторах начальные значения данных равны значениям, указанным в статье-описания-данных.)

должно быть числовым. Аргументы арифметической операции должны быть числовыми, но данное, используемое в варианте GIVING для записи результата, может быть и числовым редактируемым. Таким образом, можно, например, значение 080 сложить со значением 00262 и получить результат в виде \$3.42.

Размеры результатов могут оказаться слишком велики для запоминания в принимающем поле независимо от того, является ли оно числовым или числовым-редактируемым. В случае когда в результате слишком много цифр справа от десятичной точки, он будет усечен или округлен, и в этом обычно нет ничего страшного. Но, когда слишком много цифр расположено слева от десятичной точки,

тот факт, что отбрасываются самые значащие цифры, обычно оказывается существенным. Так, например, запоминание результата умножения 20.0 на 20.0 в данном с шаблоном 9(2)V9(2) приведет к ответу 0000. Для защиты от любых ошибочных ответов предусмотрен вариант ON SIZE ERROR (ПРИ ПЕРЕПОЛНЕНИИ). В случае его использования будет обнаруживаться переполнение значащих цифр и будут выполняться операторы, непосредственно следующие за словами ON SIZE ERROR. Эти операторы должны быть повелительными, т. е. они не должны включать никаких условных передач управления. Программисту не всегда легко решить, что же делать при таком переполнении, но он по крайней мере имеет возможность узнать об этом. Примером использования такого варианта служит оператор:

ADD A B GIVING C ON SIZE ERROR
GO TO ERROR-STOP.

Вариант ON SIZE ERROR никоим образом не исправляет и не изменяет неправильный ответ, он просто обнаруживает его. Программисту, конечно, необходимо анализировать при составлении программы, сколько потребуется цифр в принимающем данном для того, чтобы избежать переполнения. Но такая защита нужна независимо от того, как тщательно был проведен этот анализ.

Оператор ADD

Оператор ADD (СЛОЖИТЬ) служит для сложения двух или более данных и запоминания результата. В процессе выполнения оператора производится преобразование всех чисел и выравнивание относительно десятичной точки в соответствии с фразами PICTURE.

Оператор ADD имеет два формата.

Формат 1:

ADD { имя-данного-1 } { имя-данного-2 } ... TO
 литерал-1 литерал-2
 имя-данного-3 [ROUNDED] [имя-данного-4 [ROUNDED]] ...
 [ON SIZE ERROR повелительные-операторы]

Формат 2:

ADD { имя-данного-1 } { имя-данного-2 } { имя-данного-3 } ...
 литерал-1 литерал-2 литерал-3
 GIVING имя-данного-4 [ROUNDED] [имя-данного-5 [ROUNDED]] ...
 [ON SIZE ERROR повелительные-операторы]

Язык формального описания (с подчеркиванием обязательных слов, фигурными скобками для обязательной статьи, квадратными скобками для необязательных статей, расположенными друг над другом словами для альтернатив и многообразием для повторного использования) оказывается здесь очень полезным, так как дать точное определение на словах было бы затруднительно. Примеры допустимых операторов ADD (для краткости с одиобуквенными именами) приводятся ниже:

```
ADD A TO B
ADD A B C TO D ROUNDED
ADD A TO B SIZE ERROR STOP RUN
ADD A B C D E F GIVING G ROUNDED
ADD A B C GIVING D E ROUNDED F
ADD 100 TO A
ADD A B C TO D E F
```

При использовании формата 1 значения, предшествующие слову TO (C), алгебраически складываются, а затем полученная сумма прибавляется к данному с именем имя-данного-3. Затем сумма значений, предшествующих слову TO, прибавляется к данному с именем имя-данного-4 (если оно присутствует) и так ко всем остальным данным, размещенным после TO. Все данные должны быть числовыми. При использовании формата 2 значения, предшествующие слову GIVING (ПОЛУЧАЯ), алгебраически складываются и результат запоминается в данном с именем имя-данного-4 и во всех остальных данных, указанных после GIVING. Обратите внимание, что в формате 2 нет слова TO. Результат запоминается в данном с именем имя-данного-4, а не прибавляется к нему, так что эта операция оказывается замещающей. В формате 1 данное с именем имя-данного-3 (и, возможно, имя-данного-4 и т. д.) увеличивается; в формате 2 данное с именем имя-данного-4 (и т. д.) замещается. Если бы в примерах операторов, приведенных выше (где обращение происходит к данным с именами A, B, C и т. д.), начальные значения данных были бы таковы: A=1, B=2 и так далее по возрастанию до G=7, то результаты после выполнения каждого оператора были бы следующими (рассматривается не последовательное применение операторов к получающимся результатам, а изолированное выполнение операторов):

```
A=1    B=3
A=1    B=2    C=3    D=10
A=1    B=3
A=1    B=2    C=3    D=4    E=5    F=6    G=21
```

A = 1 B = 2 C = 3 D = 6 E = 6 F = 6

A = 101

A = 1 B = 2 C = 3 D = 10 E = 11 F = 12

Оператор SUBTRACT

Оператор SUBTRACT (ОТНЯТЬ) служит для вычитания одной величины или суммы двух или более величин из одного или более числовых данных. Формальное определение этого оператора аналогично определению оператора ADD.

Формат 1:

$$\text{SUBTRACT} \left\{ \begin{array}{l} \text{имя-данного-1} \\ \text{литерал-1} \end{array} \right\} \left[\begin{array}{l} \text{имя-данного-2} \\ \text{литерал-2} \end{array} \right] \dots \text{FROM}$$

$$\text{имя-данного-3} \left[\text{ROUNDED} \right] \left[\text{имя-данного-4} \left[\text{ROUNDED} \right] \right] \dots$$

$$\left[\text{ON SIZE ERROR} \text{ повелительные-операторы} \right]$$

Формат 2:

$$\text{SUBTRACT} \left\{ \begin{array}{l} \text{имя-данного-1} \\ \text{литерал-1} \end{array} \right\} \left[\begin{array}{l} \text{имя-данного-2} \\ \text{литерал-2} \end{array} \right] \dots \text{FROM}$$

$$\left\{ \begin{array}{l} \text{имя-данного-3} \\ \text{литерал-3} \end{array} \right\} \text{GIVING} \text{имя-данного-4} \left[\text{ROUNDED} \right]$$

$$\left[\text{имя-данного-5} \left[\text{ROUNDED} \right] \right] \dots$$

$$\left[\text{ON SIZE ERROR} \text{ повелительные-операторы} \right]$$

В операторе SUBTRACT особое внимание надо обратить на то, что значения данных, предшествующих слову FROM (ОТ) (в обоих форматах), суммируются перед вычитанием из одного или более данных, следующих за словом FROM. Обратите также внимание, что в формате 2 присутствует литерал-3; таким образом, значения можно вычитать из константы. Примеры операторов SUBTRACT таковы:

SUBTRACT A FROM B

SUBTRACT A B FROM C ROUNDED

SUBTRACT A FROM B GIVING C D E

SUBTRACT A FROM 100 GIVING B

SUBTRACT A B C FROM D ROUNDED E ROUNDED

ON SIZE ERROR GO TO FIX-UP-PARAGRAPH

Если, как и раньше, начальными значениями данных А, В, ... были бы 1, 2, ... , то результаты каждого оператора, выполняющегося независимо от других, были бы таковы:

$$A = 1 \quad B = 1$$

$$A = 1 \quad B = 2 \quad C = 0$$

$$A = 1 \quad B = 2 \quad C = 1 \quad D = 1 \quad E = 1$$

$$A = 1 \quad B = 99$$

$$A = 1 \quad B = 2 \quad C = 3 \quad D = -2 \quad E = -1$$

4.2. Операторы MULTIPLY и DIVIDE

В большинстве обычных экономических задач можно обойтись только операциями сложения, например в бухгалтерских расчетах или при составлении итоговых отчетов, но в областях, требующих более сложной обработки, необходимы арифметические операции умножения и деления. При употреблении этих операций необходимо заботиться о точности и масштабировании. При умножении значения данных могут превзойти установленные пределы. При делении на очень маленькое число или на нуль также может произойти подобное превышение пределов и последующее переполнение. Программист должен постоянно следить за масштабом (или положением десятичной точки).

Оператор MULTIPLY (УМНОЖИТЬ) формирует произведение двух числовых элементарных данных и запоминает результат. Умножаются только два данных; здесь нет перемножения более двух данных, как это было в операторах ADD и SUBTRACT. Формальное определение оператора MULTIPLY приведено ниже.

Формат 1:

$$\text{MULTIPLY} \left\{ \begin{array}{l} \text{имя-данного-1} \\ \text{литерал-1} \end{array} \right\} \quad \text{BY} \quad \text{имя-данного-2} \quad [\text{ROUNDED}]$$

[ON SIZE ERROR повелительные-операторы]

формат 2:

$$\text{MULTIPLY} \left\{ \begin{array}{l} \text{имя-данного-1} \\ \text{литерал-1} \end{array} \right\} \quad \text{BY} \quad \left\{ \begin{array}{l} \text{имя-данного-2} \\ \text{литерал-2} \end{array} \right\}$$

GIVING имя-данного-3 [ROUNDED] [имя-данного-4 [ROUNDED]] ...

[ON SIZE ERROR повелительные-операторы]

При использовании первого формата значение данного с именем имя-данного-1 или литерал-1 умножается на значение данного с именем имя-данного-2 и запоминается в этом последнем данном. В формате 2 умножаются два значения и ответ запоминается в данном с именем имя-данного-3. Благодаря варианту GIVING данное с именем имя-данного-3 или любое, следующее за GIVING, может быть числовым-редактируемым. Данные с именами имя-данного-1 и имя-данного-2 должны быть числовыми. Примеры операторов MULTIPLY:

```
MULTIPLY A BY B
MULTIPLY A BY 100 GIVING B
MULTIPLY A BY B GIVING C D ROUNDED E
MULTIPLY A BY B GIVING C ROUNDED ON SIZE
ERROR GO TO CHECK-NUMBERS
```

Заметьте, что следующий оператор неверен, так как литерал не может находиться во второй позиции формата 1

MULTIPLY A BY 100 (НЕВЕРНО!)

Оператор DIVIDE (РАЗДЕЛИТЬ) предназначен для деления двух чисел и запоминания частного. В каждой операции деления важно знать, что на что делится; этот вопрос разрешается с помощью ключевых слов INTO (НА) и BY (НА), но в простейшем из двух форматов используется слово INTO.

Формат 1:

$$\text{DIVIDE} \left\{ \begin{array}{l} \text{имя-данного-1} \\ \text{литерал-1} \end{array} \right\} \text{ INTO } \left\{ \begin{array}{l} \text{имя-данного-2} \\ \text{[ROUNDED]} \end{array} \right\} \dots$$

- [ON SIZE ERROR повелительные-операторы]

Значение каждого из данных, упомянутых после слова INTO, делится на значение данного с именем имя-данного-1 или на литерал-1, и результаты заносятся в соответствующие данные. Таким образом, для того чтобы разделить ряд данных пополам, достаточно выполнить оператор:

DIVIDE 2 INTO A B C D.

В более общем формате необходим вариант GIVING.

Формат 2:

$$\text{DIVIDE} \left\{ \begin{array}{l} \text{имя-данного-1} \\ \text{литерал-1} \end{array} \right\} \left\{ \begin{array}{l} \text{INTO} \\ \text{BY} \end{array} \right\} \left\{ \begin{array}{l} \text{имя-данного-2} \\ \text{литерал-2} \end{array} \right\}$$

GIVING имя-данного-3 [ROUNDED] [имя-данного-4 [ROUNDED]] ...

[REMAINDER имя-данного-5]
[ON SIZE ERROR повелительные-операторы]

Обратите внимание, что снова в первом формате данное с именем имя-данного-2 используется одновременно в качестве операнда арифметической операции и для запоминания результата. Оно должно быть числовым. Как и раньше, данное, упоминаемое после слова GIVING, может быть либо числовым, либо числовым редактируемым. Вариант ON SIZE ERROR тоже действует, как и раньше. Деление на нуль всегда приводит к ошибке независимо от масштабирования. Примеры оператора DIVIDE таковы:

DIVIDE A INTO B

DIVIDE A BY B GIVING C

С этой операцией связан новый вариант — REMAINDER (ОСТАТОК). Если используется этот вариант, то остаток от деления сохраняется и помещается в данное с именем имя-данного-5; если этот вариант опущен, то остаток не сохраняется. Остаток появляется в обоих случаях и представляет собой разность между делимым и произведением частного и делителя. Остаток вычисляется до выполнения округления (варианта ROUNDED). Когда применяется вариант REMAINDER, может использоваться только единственное данное с именем имя-данного-3, т. е. не допускается вычисление и запоминание нескольких результатов.

На рис. 4.2 показан пример раздела PROCEDURE DIVISION, использующий арифметические операции. Процедура будет считать записи файла и вычислять общий процент (TOTAL-INTEREST), суммируя отдельные проценты из записей (INDIVIDUAL-INTEREST).

Упражнения

1. Пусть в секции рабочей-памяти задано описание

WORKING-STORAGE SECTION.

01 COMPUTATIONAL-VARIABLES.

05 A-X PICTURE IS S9(5)99
 VALUE IS -3.25.

05 B-X PICTURE IS S9(5)V99
 VALUE IS 13.75.

Пример раздела PROCEDURE DIVISION.

PROCEDURE DIVISION.

START-PARAGRAPH.

MOVE ZERO TO TOTAL-INTEREST.

READ-PARAGRAPH.

READ DATA-FILE RECORD AT END GO TO PRINT-PARAGRAPH.

MULTIPLY .075 BY PRINCIPAL-VALUE GIVING INDIVIDUAL-INTEREST.

ADD INDIVIDUAL-INTEREST TO TOTAL-INTEREST ROUNDED.

GO TO READ-PARAGRAPH.

PRINT-PARAGRAPH.

WRITE OUTPUT-RECORD FROM TOTAL-INTEREST-RECORD.

STOP RUN.

(В данном примере операторы OPEN и CLOSE опущены)

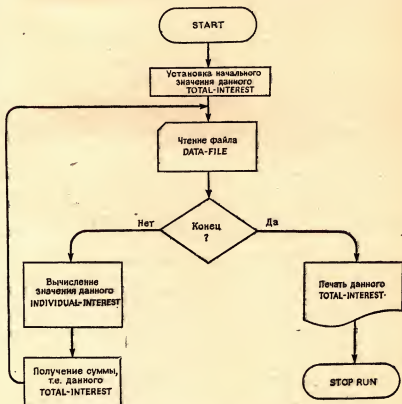


Рис. 4.2. Пример раздела процедур.

05 C-X	PICTURE IS S9 VALUE IS 4.
05 D-X	PICTURE IS S99V99 VALUE IS —11.75.

и пусть в памяти хранятся следующие значения данных:

A-X	000032 ⁵
B-X	000137 ⁵
C-X	⁴
D-X	1175

где знак числа показан над самой правой позицией. Кроме того, в рабочей-памяти задано описание:

```
01 RESULTANT-VALUES.
   05 X-X PICTURE IS S9(5)V9.
   05 Y-X PICTURE IS S9(5)V9.
   05 Z-X PICTURE IS $(2),$(3).99.
```

Напишите предложения раздела процедур, выполняющие следующие ниже операции (каждое упражнение выполняется отдельно над исходными значениями данных A-X, B-X, C-X, D-X). Кроме того, выпишите запомненные значения, которые будут получаться в результате выполнения этих предложений:

- Пусть D-X будет суммой всех элементарных данных группового данного COMPUTATIONAL-VARIABLES.
- Сложите C-X и D-X и вычтите эту сумму из A-X, а также из B-X. Поместите ответы в A-X и B-X соответственно.
- Повторите п. б, но ответы поместите в X-X и Y-X соответственно.
- Умножьте A-X на D-X и поместите ответ в Z-X.
- Разделите B-X на C-X и поместите ответ в Z-X.
- Пусть C-X будет суммой всех элементарных данных группового данного COMPUTATIONAL-VARIABLES.

2. Напишите полную КОБОЛ-программу для чтения файла подлежащих оплате счетов и образования файла записей подлежащих выплате сумм. Записи подлежащих оплате счетов таковы:

```
01 ACCOUNTS-PAYABLE-RECORD.
   05 VENDOR-NUMBER      PICTURE IS X(5).
   05 VENDOR-NAME        PICTURE IS X(15).
   05 PLANT-CODE-NO      PICTURE IS X(2).
   05 INVOICE-DUE-DATE   PICTURE IS X(6).
```

05 NUMERICAL-PART.

10 INVOICE-AMOUNT	PICTURE IS 9(4)V99.
10 DISCOUNT-PERCENT	PICTURE IS V9(3).

Выходные записи подлежащих выплате сумм должны быть таковы:

01 AMOUNT-DUE RECORD.

05 VENDOR-INFORMATION.

10 AMT-VENDOR-NUMBER	PICTURE IS X(5).
10 AMT-VENDOR-NAME	PICTURE IS X(15).
10 AMT-AMOUNT-DUE	PICTURE IS \$\$,\$\$\$,99.

Программа должна вычислить подлежащую выплате сумму для каждой записи поставщика (VENDOR) путем вычитания величины скидки из общей суммы счета (INVOICE-AMOUNT). Шесть цифр данного INVOICE-AMOUNT представляют доллары и центы; три цифры данного DISCOUNT-PERCENT (ПРОЦЕНТ-СКИДКИ) предусматривают десятые доли процента, так что 6.5% будет храниться в виде 065.

4.3. Оператор COMPUTE

Для сложных вычислений предусмотрен оператор COMPUTE (ВЫЧИСЛИТЬ), который допускает задание нескольких операций в одном операторе. В операторе COMPUTE допускаются арифметические операции сложения, вычитания, умножения и деления и, кроме того, возведения в степень. Возведение в степень позволяет находить различные степени чисел, квадратные корни из чисел. Оператор COMPUTE служит для вычислений по математическим формулам. Формальное определение оператора COMPUTE таково:

COMPUTE {имя-данного-1 [ROUNDED]} ... = арифметическое выражение
 [ON SIZE ERROR повелительные-операторы]

где данное с именем имя-данного-1 может быть либо числовым, либо числовым редактируемым, а варианты ROUNDED и ON SIZE ERROR действуют, как и раньше. Слева от знака равенства может быть указано любое число имен-данных. Арифметическое выражение представляет собой формулу, состоящую из имен-данных, литералов, круглых скобок и арифметических действий. Примеры операторов COMPUTE таковы:

```

COMPUTE A = B
COMPUTE A M G = 1.0
COMPUTE A = 5.9 + B
COMPUTE A = B + (C - D) / 3.0
  
```

Первый пример идентичен оператору MOVE; второй пример показывает использование трех имен-данных A, M и G, каждому из которых присваивается значение 1.0. « $5.9 + B$ » и « $B + (C - D) / 3.0$ » — примеры арифметических выражений. Два последних оператора заносят их значения в качестве значения данного с именем имя-данного-1.

Арифметическое выражение может быть просто именем-данного или просто литералом, как показано в первом примере. Арифметическое выражение может быть парой членов, разделенных знаком арифметической операции, причем каждый из членов может быть либо именем-данного, либо литералом, как показано в третьем примере. Арифметическое выражение также может быть рядом арифметических выражений, разделенных знаками операций. Арифметические выражения могут быть также заключены в круглые скобки. Такая возможность показана в четвертом примере. И, наконец, арифметическому выражению может предшествовать либо унарный минус (отрицание), либо унарный плюс. Все эти возможности не противоречат друг другу. Они еще раз перечислены ниже.

- а) Выражение может быть идентификатором или литералом

A или 5 или 3.46

- б) Выражение может быть парой членов (каждый из которых может быть либо идентификатором, либо литералом), разделенных знаком операции

$A + 16.9$ или $B + C$

- в) Выражение может быть рядом выражений, разделенных знаками операций

$A + 16.9 + B + C$

- г) Выражение может быть заключено в круглые скобки

$((A + 16.9) + B + (5 + C))$

- д) Выражению может предшествовать либо унарный минус, либо унарный плюс.

$-(A + B)$ или $-A$ или $+B$

Это приблизительное определение арифметических выражений, соответствующее тому, как они образуются в обычной математике. В выражениях не допускаются два знака операций, записанные подряд; знаки операций должны быть окружены пробелами. Таким образом, следующие последовательности литер не являются правильными выражениями КОБОЛа:

$A + - C$

$A - B$

(НЕВЕРНО!)

Использование круглых скобок для отделения знаков операций позволяет избежать неверных комбинаций, например $A + (-C)$.

Арифметические операции, о которых идет речь, это сложение, вычитание, умножение, деление и возведение в степень. Знаки этих операций представлены определенными символами следующим образом:

Операция	Символ	Пример	Результат
сложение	+	$5 + 3$	8
вычитание	-	$2 - 3$	- 1
умножение	*	$5 * 3$	15
деление	/	$7 / 2$	3.5
возведение в степень	**	$3 ** 2$	9

Унарный минус представлен символом вычитания и указывает на значение, противоположное по знаку. Символ возведения в степень — это пара звездочек без пробела между ними, и он не связан с символом умножения, одиночной звездочкой.

Круглые скобки управляют порядком вычислений в сложных выражениях, так что

$$(4 + 3) * 7 \text{ имеет значение } 49$$

а

$$4 + (3 * 7) \text{ имеет значение } 25$$

При отсутствии круглых скобок порядок вычислений (выполнения операций) определяется их старшинством (их рангом). Например,

$$4 + 3 * 7 \text{ имеет значение } 25$$

так как умножение делается раньше сложения, и о нем говорят, что оно имеет более высокий ранг. Старшинство рангов операций приведено ниже:

Знак операции	Ранг
унарный минус или плюс	1
**	2
* или /	3
+ или -	4

Выражение — $4 ** 2 * 3$ вычислялось бы так, как если бы круглые скобки были расставлены следующим образом:

$$(((- 4) ** 2) * 3), \text{ и привело бы к результату } 48$$

Операции умножения и деления обе имеют одинаковый ранг, и правила старшинства ничего не говорят о том, что делать в случае, когда две операции одного ранга встречаются подряд без круглых скобок. Когда последовательность выполнения не определяется ни старшинством операций, ни круглыми скобками, вычисления производятся слева направо. Например, арифметические правила не определяют последовательность вычислений в таком случае:

$$5 / 2 * 3$$

или в таком:

$$9 - 1 + 5$$

О таких выражениях говорят, что они двусмысленны и, хотя правило КОБОЛа, предписывающее вычислять слева направо, позволяет вычислить такие выражения, их следует избегать, используя круглые скобки для явного задания порядка действий, например:

$$5 / (2 * 3)$$

или

$$9 - (1 + 5)$$

Заметьте, что $-(A + 5)$ не одно и то же, что $-A + 5$.

В КОБОЛе каждый знак операции должен иметь с обеих сторон пробелы. Эти пробелы нужны для того, чтобы облегчить компилятору распознавание знаков операций и контролировать правильность выражения. Пробел может следовать за левой скобкой или непосредственно предшествовать правой скобке, но это не обязательно, и такие пробелы часто опускаются. Ниже приводятся несколько дополнительных примеров операторов COMPUTE:

COMPUTE A = - B

COMPUTE A = (3.059 * C) ** N / 1.05

COMPUTE A = 1.0 + B + B ** 2

Если доход от вложенного капитала X за N лет при ежегодной прибыли в I % вычисляется по формуле:

$$X \frac{(1 + I)^N - 1}{I}$$

то для его вычисления можно употребить оператор

COMPUTE A = X * (((1 + I) ** N - 1) / I).

Использование этого оператора во многих случаях оказывается более простым по сравнению с использованием соответствующего ему набора арифметических операторов, описанных в предыдущих разделах и часто требующих дополнительного места в рабочей-памяти для хранения промежуточных результатов. Следующие два оператора приводят к одинаковым результатам:

$$\text{COMPUTE } D-X = A-X + B-X + C-X$$

и

$$\text{ADD } A-X \ B-X \ C-X \ \text{GIVING } D-X$$

При использовании оператора COMPUTE можно употреблять только знаки арифметических операций и знак равенства. Не допускается употребление вместо знаков соответствующих слов, например MULTIPLY (УМНОЖИТЬ) или EQUALS (РАВНО). Нельзя также никак комбинировать оператор COMPUTE с арифметическими глаголами. Записывать арифметическое выражение нужно внимательно. Если используются скобки, то число левых и правых скобок должно совпадать. Все имена-данных должны быть отделены друг от друга знаками операций. Так, следующая комбинация неверна:

$$A-X(B-X + C-X) \quad (\text{НЕВЕРНО!})$$

Чтобы она стала арифметическим выражением, ее следует записать в виде

$$A-X * (B-X + C-X)$$

Точно так же произведение $A-X \ B-X$ должно быть записано как $A-X * B-X$. Другая опасность заключается в попытке возвести отрицательное число в дробную степень, так как в обычной арифметике квадратный корень из отрицательного числа не определен. Это правило применяется также к любому выражению вида

$$(-4) ** 2.5$$

Эта проблема может возникнуть и для имен-данных:

$$A-X ** B-X$$

Этот оператор привел бы к ошибке во время выполнения программы, если значения $A-X$ и $B-X$ оказались бы одновременно отрицательным и дробным соответственно.

И наконец, глагол COMPUTE не допускает вычисления функций, например логарифмов или тригонометрических функций.

Примеры

1. Площадь любого треугольника со сторонами длины A , B и C (обозначена $AREA-X$) может быть вычислена с помощью операторов:

COMPUTE $S = .5 * (A + B + C)$

COMPUTE AREA-X ROUNDED = $(S * (S - A) * (S - B) * (S - C)) ** .5$

2. Площадь круга с диаметром D (обозначена CIRCLE-AREA) вычисляется последовательностью операторов:

COMPUTE CIRCLE-AREA = $0.7854 * D ** 2$

ON SIZE ERROR

WRITE ERROR-MESSAGE-RECORD

MOVE ZERO TO CIRCLE-AREA

GO TO PARA-ONE.

Упражнения

1. Арифметическое выражение может быть построено из имен-данных и литералов за ряд шагов, использующих основные правила разд. 4.3. Например, выражение

$$-(A + 16.9) * C$$

может быть построено с помощью следующих основных правил:

Правило (а) $A \quad 16.9 \quad C$

Правило (б) $A \quad + 16.9 \quad C$

Правило (г) $(A + 16.9) \quad C$

Правило (д) $-(A + 16.9) \quad C$

Правило (в) $-(A + 16.9) * C$

Попытайтесь найти такие цепочки правил для построения следующих выражений:

а. $A + B + C$

б. $A * (B - C)$

в. $-A - B$

г. $-A * B ** C$

2. Верны ли следующие выражения?

а. $(-(A - (B - C)))$

б. $A ** -5$

в. $A / B / C / D$

г. DIVIDE A INTO B

д. $-A ** B * C$

3. Пусть P — сумма, на которую начисляются проценты, и го-довой процент равен i (в десятичном выражении). И пусть эта сумма вложена на период в n лет. Тогда напишите процедурные сегмен-ты для вычисления суммы, накопившейся в результате начисления простых процентов; суммы, накопившейся в результате ежегодного

начисления сложных процентов; суммы накопившейся в результате начисления сложных процентов s раз в год. Соответствующие формулы таковы:

$$A = P (1 + ni)$$

$$A = P (1 + i)^n$$

$$A = P (1 + i/s)^{ns}$$

4.4. Управление последовательностью выполнения

Обычная порядковая последовательность выполнения операторов КОБОЛа может быть изменена определенными управляющими операторами. Простейший из них это оператор STOP (ОСТАНОВИТЬ). Этот оператор прерывает выполнение программы либо окончательно, либо временно. В последнем случае для продолжения выполнения программы требуется вмешательство оператора вычислительной системы. Общий формат оператора STOP таков:

$$\underline{\text{STOP}} \left\{ \frac{\text{RUN}}{\text{литерал}} \right\}$$

Точно так же, как и для многих операторов КОБОЛа, эти два варианта имеют два различных смысла. Оператор STOP RUN (ОСТАНОВИТЬ РАБОТУ) приводит к окончательному прекращению работы программы, после чего ни один оператор этой программы не может быть выполнен. В последовательности повелительных операторов оператор STOP RUN должен быть записан последним:

```

READ IN-FILE RECORD AT END
CLOSE IN-FILE
WRITE FINAL-OUTPUT-RECORD
CLOSE OUTPUT-FILE
STOP RUN.
```

Любой файл, который был ранее открыт как входной или как выходной, должен быть закрыт до выполнения оператора STOP RUN. Оператор STOP литерал приводит к временному прекращению выполнения программы и к сообщению значения литерала оператору машины. Оператор с пульта управления машиной может вызвать продолжение выполнения программы. В этом случае будет выполняться оператор, непосредственно следующий в программе за оператором STOP. Значение литерала может быть как нечисловым, так и числовым:

```
STOP 12345.
```

```
STOP "ABCDEF".
```

```
STOP "CHANGE FORMS AND RESTART".
```

Окончательный останов программы, использующий формат STOP RUN, это обычный способ использования этого оператора. Допускать вмешательство человека в процедуру обработки следует только в особых случаях.

Другой способ изменения обычной последовательности выполнения заключается в применении оператора GO TO (ПЕРЕЙТИ К), который передает управление в указанную в нем точку КОБОЛ-программы. Формат оператора GO TO таков:

GO TO имя-процедуры

где имя-процедуры означает либо имя-параграфа, либо имя-секции. Если это имя-секции, то слово SECTION (СЕКЦИЯ) не появляется, а указывается только само имя этой секции. Оператор GO TO должен быть последним в последовательности операторов, так как следующим будет выполняться оператор, расположенный вслед за именем-процедуры. Примеры:

GO TO FIX-UP-NUMBERS.

GO TO INITIALIZATION-PARAGRAPH.

Оператор GO TO вызывает безусловную передачу управления.

Оператор IF (ЕСЛИ) также вызывает изменение обычной последовательности выполнения программы. Оператор IF определяется так:

$$\text{IF условие} \quad \left\{ \begin{array}{l} \text{оператор-1} \\ \text{NEXT SENTENCE} \end{array} \right\} \left[\text{ELSE} \left\{ \begin{array}{l} \text{оператор-2} \\ \text{NEXT SENTENCE} \end{array} \right\} \right]$$

Как оператор-1, так и оператор-2 могут быть любыми операторами КОБОЛа или любой последовательностью операторов КОБОЛа с тем ограничением, что условный оператор должен быть последним в ряду операторов. Условие будет описано более полно в следующем разделе, а сейчас о нем можно сказать, что это описание ситуации, которая во время выполнения программы может быть либо истинной, либо ложной. Ниже приводятся примеры операторов IF:

IF A IS POSITIVE MOVE ZERO TO A ELSE COMPUTE A = 3 * B.
IF TERMINAL-SIGNAL IS EQUAL TO 999 GO TO P-10.

При выполнении оператора IF производятся следующие действия:

- 1) если условие истинно, то выполняется оператор-1 и управление передается следующему предложению в обход оператора-2, если он присутствует;
- 2) если условие ложно, то выполняется оператор-2, если он присутствует, а оператор-1 обходится.

Оператор-1 и оператор-2 могут быть любыми операторами. Они могут быть и операторами IF. Используя эту возможность, можно составить чрезвычайно сложные предложения, например:

```
IF A IS ZERO IF B IS ZERO IF C IS ZERO  
WRITE A-B-C ELSE WRITE A-B ELSE WRITE A  
ELSE STOP RUN.
```

Вероятно, до тех пор пока не будет накоплен достаточный опыт использования этих операторов, лучше всего применять простые и ясные операторы IF. При выполнении фразы NEXT SENTENCE (СЛЕДУЮЩЕЕ ПРЕДЛОЖЕНИЕ) управление передается предложению, следующему за завершающей точкой. Необязательная фраза ELSE (ИНАЧЕ) необходима для образования пар IF-ELSE, если операторы IF включены либо в оператор-1, либо в оператор-2. Подобные сложности более подробно описываются в гл. 8; поэтому пока следует использовать оператор IF в простейшей форме:

IF условие повелительные-операторы

4.5. Условия

Арифметические выражения порождают результаты, которые можно запоминать в памяти машины. Существует другой тип результатов, вырабатываемых КОБОЛ-программой. Результат этого типа может быть вычислен, но его нельзя запомнить, однако его можно *проверить*. Такой результат называется *условием*. Он может принимать только одно из двух возможных значений: *истина* или *ложь*. Условие — это последовательность слов КОБОЛа, описывающая ситуацию, которая во время выполнения программы может быть проверена, т. е. она может быть либо истинной, либо ложной. Никакие другие возможности, кроме истины или лжи, не допускаются. Например, в случае условия:

```
JOB-NUMBER IS EQUAL TO ZERO  
(НОМЕР-РАБОТЫ РАВЕН НУЛЬ)
```

во время выполнения программы существуют только две возможности: либо текущее значение данного JOB-NUMBER в точности равно нулю, либо нет. В первом случае значением условия JOB-NUMBER IS EQUAL TO ZERO является истина, во втором случае — ложь. Для другого условия

```
JOB-NUMBER IS GREATER THAN 888  
(НОМЕР-РАБОТЫ БОЛЬШЕ 888)
```

в любом случае результатом является либо истина (значение данного JOB-NUMBER больше, чем 888), либо ложь (значение данного

JOB-NUMBER или равно 888, или меньше, чем 888). Условия необходимы для того, чтобы во время выполнения программы иметь возможность действовать в зависимости от этих условий, используя их в операторе IF для управления последовательностью выполнения, например:

IF JOB-NUMBER IS EQUAL TO ZERO GO TO
CALCULATE-RESULTS ELSE CLOSE OUT-FILE
STOP RUN.

В КОБОЛе допускаются несколько типов условий. Из отдельных условий можно формировать более сложные условия. Изначально условие определяется как условие-отношения, условие-класса или условие-знака. Из этих условий можно составлять более сложные условия по следующим правилам. Условие — это любое условие с предшествующим ключевым словом NOT (НЕ). Условие может быть также образовано из любых двух условий, разделенных одним из ключевых слов AND (И) или OR (ИЛИ). Это описание подобно описанию арифметического выражения, в котором нечто определяется посредством самого себя. Такое определение называется рекурсивным определением. Таким образом, говоря, что условие — это условие, которому предшествует слово NOT, мы даем циклическое, но правильное определение. Далее, условие может быть условием, заключенным в круглые скобки. Опять же это определение можно применять повторно. Условие может быть таким:

JOB-NUMBER IS EQUAL TO ZERO

а так как определение допускает заключение в круглые скобки, то
(JOB-NUMBER IS EQUAL TO ZERO)

также является условием. Повторно применяя это определение, получаем следующие условия:

((JOB-NUMBER IS EQUAL TO ZERO))
(((JOB-NUMBER IS EQUAL TO ZERO)))

Использование рекурсивного определения позволяет строить условия из условий. Так JOB-NUMBER IS EQUAL TO ZERO — это условие-отношения. Условие-отношения является условием. Точно так же, условием является фраза

NOT JOB-NUMBER IS EQUAL TO ZERO

так как это условие с предшествующим ключевым словом NOT. Комбинация двух условий с соединительным OR — это также условие, например:

NOT JOB-NUMBER IS EQUAL TO ZERO OR
ANOTHER-ITEM IS GREATER THAN 400

Значением такой фразы, представляющей в целом некоторое условие, является либо истина, либо ложь независимо от того, из скольких подчиненных условий она составлена. Обратите внимание, что круглые скобки могут изменить смысл, например:

NOT (JOB-NUMBER IS EQUAL TO ZERO OR
ANOTHER-ITEM IS GREATER THAN 400)

Повторим еще раз формальное определение условия. Условие это:

условие-отношения	или
условие-знака	или
условие-класса	или
NOT условие	или
(условие)	или
условие AND условие	или
условие OR условие	или

Условие-отношения, условие-знака и условие-класса будут определены ниже. Начинающему программисту во избежание логических неопределенностей всегда следует заключать в скобки условие, которому предшествует слово NOT. Единственное ограничение на это правило заключается в том, что не допускается несколько подряд идущих слов NOT. Ниже приведены примеры отдельных условий:

PAY-AMOUNT IS NEGATIVE
CARD-IDENTIFICATION IS ALPHABETIC
NOT (RESULT-X IS EQUAL TO ANSWER-Y)
A IS GREATER THAN B OR C IS EQUAL TO ZERO
3 * A + 9.5 IS POSITIVE

Условие-отношения

Условие-отношения вызывает сравнение двух величин, каждая из которых может быть значением литерала, имени-данного или арифметического выражения. Арифметические выражения всегда имеют числовые значения, а литерал и имя-данного могут иметь нечисловые значения. Формальное определение условия отношения таково:

$\left\{ \begin{array}{l} \text{имя-данного-1} \\ \text{литерал-1} \\ \text{арифметическое-выражение-1} \end{array} \right\}$	IS	[NOT]	GREATER THAN	$\left\{ \begin{array}{l} \text{имя-данного-2} \\ \text{литерал-2} \\ \text{арифметическое-выражение-2} \end{array} \right\}$
	IS	[NOT]	LESS THAN	
	IS	[NOT]	EQUAL TO	

Одновременно по обе стороны отношения не могут находиться литералы, так что не допускаются условия типа:

3.45 IS GREATER THAN 1.56
(НЕВЕРНО!)

Примеры допустимых условий-отношения таковы:

A IS GREATER THAN 55

(3 — 5 * A) IS LESS THAN B — C

346 IS NOT GREATER THAN DAY-OF-YEAR

Заметьте, что операция отношения равенства обозначается EQUAL TO (РАВНО), использование слова EQUALS не допускается. Такая ошибка является типичной, и следующее условие-отношения не-правильно:

A-ITEM EQUALS B-ITEM (НЕВЕРНО!)

Кроме того существует два различных слова NOT (НЕ): одно используется в условии-отношения и влияет на сравнение, а другое было определено для любого условия (это то NOT, которое предшествует условию). Часто эти два ключевых слова NOT одинаково влияют на условие в целом, так что условия:

A IS NOT GREATER THAN 55

и

NOT (A IS GREATER THAN 55)

оба истинны или ложны одновременно. В первом случае условие истинно для значений A, меньших или равных 55, и ложно для всех значений, больших 55. Во втором случае внутреннее условие ложно для значений, меньших или равных 55, но результат меняется на противоположный в результате применения внешнего NOT, и условие в целом становится истинным. Те же рассуждения применимы и в случае, когда значения A больше 55. Однако в более сложных ситуациях эти разные NOT могут по-разному влиять на значение условия в целом, например:

NOT (A IS GREATER THAN 55 OR B IS LESS THAN 35)

в сравнении с

A IS NOT GREATER THAN 55 OR B IS NOT LESS THAN 35

На рис. 4.3 представлены две таблицы истинности, в которых приведены значения этих двух условий для различных значений данных A и B. В последних столбцах каждой таблицы приведены значения истинности для двух полных условий, и они не одинаковы. В случае когда данное A равно 55, а данное B равно 34, первое условие принимает значение ложь, в то время как второе условие принимает значение истина. При A=56 и B=34 оба условия будут иметь значение ложь, однако, чтобы они были эквивалентными, необходимо, чтобы результаты совпадали для всех комбинаций значений данных. При составлении этих таблиц учитывалось, что влия-

Первое условие: NOT (A IS GREATER THAN 55 OR B IS LESS THAN 35)

Значения данных		A IS GREATER THAN 55	B IS LESS THAN 35	A-отношение OR B-отношение	NOT (условие)
A	B				
55	34	ложь	истина	истина	ложь
56	34	истина	истина	истина	ложь
55	35	ложь	ложь	ложь	истина
56	35	истина	ложь	истина	ложь

Второе условие: A IS NOT GREATER THAN 55 OR B IS NOT LESS THAN 35

Значения данных		A IS NOT GREATER THAN 55	B IS NOT LESS THAN 35	A-отношение OR B-отношение
A	B			
55	34	истина	ложь	истина
56	34	ложь	ложь	ложь
55	35	истина	истина	истина
56	35	ложь	истина	истина

Рис. 4.3. Сравнение двух условий.

ние отрицания NOT заключается в изменении на противоположное значения истинности условия, модифицируемого этим отрицанием, а соединительное OR приводит к истинному результату, если одно или оба из связываемых им условий имеют значение истина. Влияние отрицания NOT зависит от остальной части условия. Обратите внимание, что в результате замены в первом из двух рассматриваемых примеров союза OR на союз AND условия становятся эквивалентными, так как AND приводит к истинному результату, только когда оба соединяемые им условия истинны. Так что условие NOT (A IS GREATER THAN 55 AND B IS LESS THAN 35)

для всех комбинаций значений А и В принимает те же значения истинности, что и условие

A IS NOT GREATER THAN 55 OR B IS NOT LESS THAN 35

Для величин, относящихся к числовой категории, основой для сравнения в условии-отношения являются их алгебраические значения. Если одна из двух или обе величины являются нечисловыми, то осуществляется полтерное сравнение, начиная слева. При сравнении литер буква В считается больше, чем буква А, так что понятие «больше, чем» вводится и для букв в соответствии с алфавитным порядком. Два значения JOHN SMITH и JANE SMITH сравниваются на основе буквенного сравнения и всегда считается, что первое больше. Если длины двух сравниваемых данных не равны, тогда более короткое дополняется пробелами (только для сравнения) до более длинного.

Условие-знака

Условие-знака проверяет либо знак числа, либо равенство числа нулю. Фактически условие-знака определяет, является ли числовое данное большим, чем нуль (положительным), равным нулю (неположительным и неотрицательным) или меньшим, чем нуль (отрицательным). Проверять литералы на знак нельзя, и все имена-данных должны ссылаться на числовые данные. Общий формат таков:

$$\left\{ \begin{array}{l} \text{имя-данного} \\ \text{арифметическое-выражение} \end{array} \right\} \text{ IS } [\text{NOT}] \left\{ \begin{array}{l} \text{POSITIVE} \\ \text{ZERO} \\ \text{NEGATIVE} \end{array} \right\}$$

Примеры условий-знака:

A IS POSITIVE

3 * 5 — A NOT POSITIVE

PAY-AMOUNT IS NEGATIVE

Заметьте, что сравнение с нулем можно выполнить двумя способами: либо с помощью условия-отношения

A IS EQUAL TO ZERO

либо с помощью условия-знака

A IS ZERO

Существует различие между смыслом слов POSITIVE (ПОЛОЖИТЕЛЬНО) и NOT NEGATIVE (НЕ ОТРИЦАТЕЛЬНО). Положительность не включает равенство нулю, а неотрицательность включает. Так же как и для всех условий, значением условия-знака может быть или истина, или ложь.

Условие-класса

Условие-класса определяет, является ли значение данного числовым (составленным только из цифр и, может быть, знака) или буквенным (составленным только из букв и пробелов). Определение условия-класса таково:

$$\text{имя-данного IS [NOT] } \left\{ \begin{array}{l} \text{NUMERIC} \\ \text{ALPHABETIC} \end{array} \right\}$$

Условие-класса в основном следует использовать только для проверки класса буквенно-цифровых данных. Однако можно использовать его и для числовых данных, но только в том случае, если употребляется ключевое слово NUMERIC (ЧИСЛОВОЕ). Это условие можно использовать также и для буквенных данных, но только в том случае, если употребляется слово ALPHABETIC (БУКВЕННОЕ). Так как буквенно-цифровые данные могут содержать любую литеру, то для них подходят оба ключевых слова. Для того чтобы условие-класса вида

PAY-NUMBER IS NUMERIC

было истинным, каждая литера данного PAY-NUMBER должна быть цифрой. Точно так же, класс ALPHABETIC ограничен только буквами и пробелами. Примеры допустимых условий-класса таковы:

ALPHA-DATE IS NOT ALPHABETIC
RECORD-X IS ALPHABETIC

В условии-класса могут встречаться только имена-данных; нн литералы, нн арифметические выражения не допускаются.

Использование ключевых слов AND, OR, NOT

Как определено ранее, любому условию может предшествовать ключевое слово NOT и любые два условия могут быть разделены ключевыми словами AND или OR. Этим ключевым словам должен предшествовать и за ними должен следовать по крайней мере один пробел точно так же, как и для знаков арифметических операций. Смысл этих знаков логических операций примерно отражается обычным смыслом соответствующих им ключевых слов. Если некоторое условие истинно, то такое же условие со словом NOT впереди ложно. Если условие записано в виде

условие-1 AND условие-2

то для того, чтобы оно было истинным, необходимо, чтобы и условие-1, и условие-2 были истинными. Если хотя бы одно из них ложно, то ложно условие в целом. Логическую операцию AND можно мыслить как операцию *все*: все составляющие условия должны быть истинными для того, чтобы их комбинация была истинной. Логическая операция OR подобна операции *любой*: если любое из составляющих условий истинно, то истинно и условие в целом. Примеры таких условий:

A GREATER THAN B AND B EQUAL TO 15
 TOTAL-HOURS IS GREATER THAN 40 OR DAY-OF-WEEK IS
 EQUAL TO "SAT" OR HOUR-OF-DAY GREATER THAN 1800
 RESULT-X IS LESS THAN 35 AND RESULT-X IS GREATER
 THAN 30

Примеры

1. Решение квадратного уравнения вида

$$ax^2 + bx + c = 0$$

дается формулой

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Это решение мнимое, если величина ($b^2 - 4ac$) отрицательна, так как квадратный корень из отрицательного числа в алгебре действительных чисел не допускается. Решение на КОБОЛе могло бы быть записано в виде:

P-1.

```
COMPUTE RADICAL-X = B ** 2 - 4 * A * C.
IF RADICAL-X IS NEGATIVE
  MOVE "ANSWER IS IMAGINARY" TO MESSAGE-X
  WRITE PRINT-LINE
  GO TO P-10.
COMPUTE ROOT-ONE = (- B + RADICAL-X) / (2 * A).
COMPUTE ROOT-TWO = (- B - RADICAL-X) / (2 * A).
```

2. Оценку данного DATA-ITEM можно было бы выполнить с помощью следующей процедуры:

P-1.

```
READ INPUT-FILE RECORD AT END GO TO P-10.
IF DATA-ITEM IS NOT NUMERIC
  OR DATA-ITEM IS NEGATIVE
  OR DATA-ITEM IS GREATER THAN 5800
  GO TO P-9.
```

3. Для управления процессом ввода записей можно было бы использовать числовой код CODE-ITEM в виде цифры, размещающийся в записи:

P-1.

```
READ DATA-FILE RECORD AT END GO TO P-10.  
IF CODE-ITEM IS EQUAL TO 1  
  GO TO PROCESS-RECORD-TYPE-ONE.  
IF CODE-ITEM IS EQUAL TO 2  
  GO TO PROCESS-RECORD-TYPE-TWO.  
GO TO NO-SUCH-RECORD.
```

Упражнения

1. Используя любые из трех основных условий (условие-отношения, условие-знака или условие-класса), составьте более сложные условия с помощью ключевого слова NOT, круглых скобок и знаков логических операций. Составьте четыре таких примера.

2. Нарисуйте блок-схему следующего оператора IF, принимая в каждом ромбе блок-схемы только одно решение:

P-1.

```
IF  
NOT (A IS NOT LESS THAN 50 AND B IS ZERO)  
OR A + B IS EQUAL TO 3  
OR (ALPHA-X IS NOT ALPHABETIC AND  
    C IS NOT NEGATIVE)  
GO TO P-3.
```

P-2.

...

3. Правильны ли эти условия?

а. A IS NOT EQUAL TO ZERO

б. 10 IS GREATER THAN 9

в. $A - 3 * B$ IS NEGATIVE

г. A IS LESS THAN ZERO

д. $G + 4$ EQUALS $3 * A - B$

е. ALPHA-X IS EQUAL TO "ABCD"

ж. A AND B OR C

з. RECORD-X IS NUMERIC AND POSITIVE

4.6. Повторяющиеся процедуры

Одним из методов, используемых при написании программ решений задач, является организация циклов. С помощью циклов процедура пишется один раз, а при выполнении программы многократно повторяется. Применение циклов не экономит машинного времени, но зато оно экономит время при программировании и уменьшает размер программы. Очень простой пример этого понятия показан с помощью сравнения двух программных сегментов, считывающих файл, содержащий точно пять записей.

Сегмент 1:

LOOP-X.

```
READ INPUT-FILE RECORD AT END STOP RUN.  
READ INPUT-FILE RECORD AT END STOP RUN.  
READ INPUT-FILE RECORD AT END STOP RUN.  
READ INPUT-FILE RECORD AT END STOP RUN.  
READ INPUT-FILE RECORD AT END STOP RUN.
```

NEXT-X.

Сегмент 2:

LOOP-X.

```
READ INPUT-FILE RECORD AT END GO TO NEXT-X.  
GO TO LOOP-X.
```

NEXT-X.

Вообще программный цикл состоит из четырех частей:

- а) часть начальной установки,
- б) часть модификации,
- в) часть исполнения,
- г) часть контроля выхода.

Простой пример пяти операторов READ не включает все части цикла только потому, что файл INPUT-FILE содержит точно пять записей.

Связь между этими частями можно увидеть из примеров блок-схем, приведенных на рис. 4.4. Вход в собственно цикл осуществляется через часть начальной установки, но остальные три части выполняются в цикле и могут располагаться в любом порядке, как показано в двух приведенных примерах. Часть начальной установки используется для установки исходного состояния, которое будет модифицироваться в части модификации и проверяться в части контроля выхода. Эти три части являются частями управления циклом и определяют число фактических выполнений части исполнения. Эти управляющие части представляют собой накладные расходы на

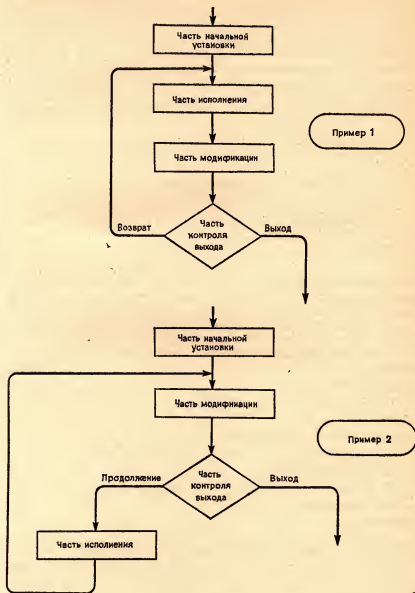


Рис. 4.4. Две блок-схемы, показывающие возможные связи между частями цикла.

организацию цикла и были бы не нужны, если бы цикл был раскручен и часть исполнения была бы написана подряд нужное число раз. Состояние, которое устанавливается, модифицируется и контролируется, может представлять собой данное, используемое в качестве счетчика для определения числа выполнений основной процедуры. Предположим, что в определении данного BLANK-LINE (ПУСТАЯ СТРОКА) указана фраза VALUE IS ALL SPACES (ЗНАЧЕНИЕ ПРОБЕЛЫ). Тогда следующий сегмент распечатает десять пустых строк:

```
MOVE ZERO TO LINE-COUNTER-X.  
LOOP-X.  
WRITE PRINT-RECORD FROM BLANK-LINE.  
ADD 1 TO LINE-COUNTER-X.  
IF LINE-COUNTER-X IS LESS THAN 10  
GO TO LOOP-X.  
CONTINUE-X.
```

Это расположение четырех частей в точности соответствует первому примеру на рис. 4.4. Размещение частей модификации и контроля перед частью исполнения, как показано во втором примере на рис. 4.4, называется *методом предварительного анализа выхода из цикла*, и некоторые программисты предпочитают именно его. Изменение взаимного расположения управляющих частей цикла требует изменения самих этих частей, в чем убеждает нижеприведенный кусок программы:

```
MOVE ZERO TO LINE-COUNTER-X.  
LOOP-X.  
ADD 1 TO LINE-COUNTER-X.  
IF LINE-COUNTER-X IS GREATER THAN 10  
GO TO CONTINUE-X.  
WRITE PRINT-RECORD FROM BLANK-LINE.  
GO TO LOOP-X.  
CONTINUE-X.
```

Необходимость задания фактического числа повторений части исполнения цикла на первый взгляд заставляет программистов, сидя за своими письменными столами, старательно считать на пальцах, какое число повторений требуется в каждом отдельном случае. С другой стороны, заменив литерал, определяющий число повторений (10 в приведенном выше примере), на имя-данного, например на LINE-LIMIT, мы получим, что число повторений будет определяться значением данного LINE-LIMIT. Засылая в данное LINE-

LIMIT определенное число перед выполнением цикла, мы напечатаем любое требуемое число пустых строк.

Цикл можно применить еще и для чтения файла до тех пор, пока не будет обнаружена определенная запись, т. е. пока не будет выполнено определенное условие:

```
OPEN INPUT CARD-FILE.  
LOOP-X. READ CARD-FILE AT END STOP RUN.  
        IF STOCK-NUMBER IS EQUAL TO 505138  
          GO TO PROCESS-RECORD-PARAGRAPH  
        ELSE GO TO LOOP-X.
```

Заметьте, что оператор IF можно было бы записать так:

```
LOOP-X. READ CARD-FILE AT END STOP RUN.  
        IF STOCK-NUMBER IS NOT EQUAL TO 505138  
          GO TO LOOP-X.  
PROCESS-RECORD-PARAGRAPH.
```

Вполне возможно использовать внутри циклов другие циклы подобно тому, как в примере вычисления значения

$$z = 2x + y$$

для комбинаций x и y при x , меняющемся от 0 до 10 с шагом 0.5, и y , меняющемся от 0 до 6 с шагом 2:

```
MOVE 0 TO VARIABLE-X.  
LOOP-X. MOVE 0 TO VARIABLE-Y.  
LOOP-Y. COMPUTE VARIABLE-Z = 2 * VARIABLE-X +  
        VARIABLE-Y.  
        WRITE OUTPUT-RECORD FROM RESULTS-  
        RECORD.  
        ADD 2 TO VARIABLE-Y.  
        IF VARIABLE-Y IS NOT GREATER THAN 6  
          GO TO LOOP-Y.  
        ADD 0.5 TO VARIABLE-X.  
        IF VARIABLE-X IS NOT GREATER THAN 10  
          GO TO LOOP-X.  
CONTINUE-X.
```

4.7. Примеры задач с решениями

Предположим, что во всех программах из примеров содержатся следующие разделы:

IDENTIFICATION DIVISION.

PROGRAM-ID. SAMPLE.

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER. UNIVAC-418-III.

OBJECT-COMPUTER. UNIVAC-418-III.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

SELECT INPUT-FILE ASSIGN TO CARD-READER-EIGHTY.

SELECT OUT-FILE ASSIGN TO PRINTER.

DATA DIVISION.

FILE SECTION.

FD INPUT-FILE

LABEL RECORDS ARE STANDARD.

01 INPUT-RECORD.

05 JOB-ID PICTURE IS 9(6).

05 NUMBER-X PICTURE IS S9(5)V9.

FD OUT-FILE

LABEL RECORDS ARE STANDARD.

01 OUTPUT-RECORD.

05 OUTPUT-ANSWER PICTURE IS +(10).9(2).

05 FILLER PICTURE IS X(100).

WORKING-STORAGE SECTION.

01 ANSWER-RECORD.

05 JOB-ID-RANGE PICTURE IS 9(6).

05 FILLER PICTURE IS X(100) VALUE IS
SPACE.

01 RESULT-RECORD.

05 MEAN-VALUE PICTURE IS S9(10)V99.
05 FILLER PICTURE IS X(100) VALUE IS
SPACE.

01 AVERAGE-RECORD.

05 MOVING-AVERAGE PICTURE IS S9(6)V99.
05 FILLER PICTURE IS X(100) VALUE
IS SPACE.

01 MESSAGE-RECORD.

05 ZERO-INSERT PICTURE IS 99 VALUE IS ZERO.
05 ERROR-X PICTURE IS X(33)
VALUE IS "A DIVIDE BY SMALL NUMBER OCCURRED".

01 INTERMEDIATE-RESULTS.

05 HIGHEST-JOB-ID PICTURE IS 9(6).
05 LOWEST-JOB-ID PICTURE IS 9(6).
05 NUMBER-OF-RECORDS PICTURE IS 9(8).
05 A PICTURE IS S9(5)V9.
05 B PICTURE IS S9(5)V9.
05 C PICTURE IS S9(5)V9.
05 D PICTURE IS S9(5)V9.

Пример А

Напишите раздел процедур, который будет считывать все записи входного файла и распечатывать диапазон изменения значений данного JOB-ID, т. е. разность между наибольшим и наименьшим значением (JOB-ID-RANGE).

Пример В

Напишите раздел процедур, который будет считывать все записи (независимо от их числа; при этом записей может не быть совсем) и вычислять среднее всех значений данных NUMBER-X (MEAN-VALUE). Среднее — это сумма всех значений, деленная на число значений.

Пример С

Напишите процедуру, которая будет вычислять смещающееся среднее значений данных NUMBER-X из четырех соседних записей (MOVING-AVERAGE) (т. е. среднее значение данных NUMBER-X

из записей 1—4, затем из записей 2—5, затем 3—6, 4—7 и т. д., работая со скользящей последовательностью четырех записей).

Для помощи при анализе этих примеров задач чрезвычайно рекомендуется использовать блок-схемы.

Решение А

PROCEDURE DIVISION.

START-X.

OPEN INPUT INPUT-FILE.

MOVE ZERO TO HIGHEST-JOB-ID.

MOVE 999999 TO LOWEST-JOB-ID.

READ-LOOP.

READ INPUT-FILE RECORD

AT END CLOSE INPUT-FILE GO TO PRINT-RANGE.

IF

JOB-ID IS GREATER THAN HIGHEST-JOB-ID
MOVE JOB-ID TO HIGHEST-JOB-ID.

IF

JOB-ID IS LESS THAN LOWEST-JOB-ID
MOVE JOB-ID TO LOWEST-JOB-ID.

GO TO READ-LOOP.

PRINT-RANGE.

OPEN OUTPUT OUT-FILE.

SUBTRACT LOWEST-JOB-ID FROM HIGHEST-JOB-ID
GIVING JOB-ID-RANGE.

WRITE OUTPUT-RECORD FROM ANSWER-RECORD.

CLOSE OUT-FILE.

STOP RUN.

Решение В

PROCEDURE DIVISION.

START-X.

MOVE ZERO TO NUMBER-OF-RECORDS MEAN-VALUE.

OPEN INPUT INPUT-FILE OUTPUT OUT-FILE.

READ-LOOP.

READ INPUT-FILE RECORD AT END GO TO COMPUTE-MEAN.

ADD 1 TO NUMBER-OF-RECORDS.

ADD NUMBER-X TO MEAN-VALUE.

GO TO READ-LOOP.

COMPUTE-MEAN.

DIVIDE NUMBER-OF-RECORDS INTO MEAN-VALUE
ROUNDED ON SIZE ERROR GO TO PRINT-
ERROR-MESSAGE.

WRITE OUTPUT-RECORD FROM RESULT-RECORD.

FINISH-X.

CLOSE INPUT-FILE OUT-FILE.

STOP RUN.

PRINT-ERROR-MESSAGE.

WRITE OUTPUT-RECORD FROM MESSAGE-RECORD.
GO TO FINISH-X.

Решение C

PROCEDURE DIVISION.

START-X.

OPEN INPUT INPUT-FILE.

OPEN OUTPUT OUT-FILE.

MOVE +99999.9 TO A B C D.

READ-LOOP.

READ INPUT-FILE RECORD AT END CLOSE INPUT-
FILE OUT-FILE STOP RUN.

MOVE C TO D.

MOVE B TO C.

MOVE A TO B.

MOVE NUMBER-X TO A.

IF D IS EQUAL TO +99999.9 GO TO READ-LOOP.

COMPUTE MOVING-AVERAGE = $(A + B + C + D) / 4$.

WRITE OUTPUT-RECORD FROM AVERAGE-RECORD.
GO TO READ-LOOP.

Упражнения

1. Напишите программу для чтения одной записи из файла CARD-IMAGE-FILE, содержащей числовое целое данное с именем NUMBER-OF-RECORDS-X. Считывайте последовательные записи из другого файла SEARCH-FILE до тех пор, пока не будет считано столько записей, каково значение этого целого числа, а затем распечатайте последнюю считанную запись. Таким образом, если данное NUMBER-OF-RECORDS-X равно 42, то распечатать надо сорок вторую запись файла SEARCH-FILE. Длина каждой записи файла SEARCH-FILE равна ста литерам. Описания этих двух файлов таковы:

FD SEARCH-FILE

LABEL RECORDS ARE STANDARD.

01 SEARCH-FILE-RECORD PICTURE IS X(100).

FD CARD-IMAGE-FILE.

LABEL RECORDS ARE STANDARD.

01 CARD-IMAGE-FILE-RECORD.

02 NUMBER-OF-RECORDS-X PICTURE IS 9(6).

02 FILLER PICTURE IS X(74).

2. Напишите процедурный сегмент для чтения файла MASTER-FILE, чьи записи длиной 450 позиций содержат в первых пяти позициях числовое поле. Шаблон этого поля — X(5), но цифры означают доллары и центы. Преобразуйте эти поля в числовую категорию и вычислите среднее арифметическое всех полученных значений. Среднее арифметическое — это сумма всех значений, деленная на число этих значений.

3. Измените предыдущую задачу для вычисления трех различных средних: одно — для всех значений, меньших, чем \$50, второе — для значений, больших или равных \$50 и меньших, чем \$500, и третье — для всех больших значений. Не забудьте перед вычислением присвоить всем суммам нулевые значения.

4. Имеются три файла (FILE-ONE, FILE-TWO и FILE-THREE), форматы записей которых подобны следующему:

01 RECORD-LAYOUT-ONE.

05 KEY-FIELD-ONE PICTURE IS X(10).

05 FILLER PICTURE IS X(460).

Предварительно каждый файл был упорядочен в соответствии с данным KEY-FIELD. Напишите полную КОБОЛ-программу для слияния этих файлов в один файл MASTER-FILE, в котором все записи должны быть упорядочены по значению данных KEY-FIELD. Та-

ким образом, рассматривая только поле ключа (KEY-FIELD) записей для следующих значений таких полей:

FILE-ONE	FILE-TWO	FILE-THREE
A	ADD	BAD
B	BUT	D
BA	ZED	DA
BD		

получим в файле MASTER-FILE следующую последовательность:

A, ADD, B, BA, BAD, BD, BUT, D, DA, ZED

Глава 5. Примеры программ и упражнения

5.1. Обработка файлов

В предыдущих четырех главах описывались основные правила и операторы КОБОЛа, которые представлены на рис. 5.1. Этот рисунок может быть использован в качестве справочного пособия. Но он не может служить точным описанием ядра КОБОЛа.

Программа на КОБОЛе всегда содержит четыре раздела, которые должны следовать в указанном порядке. В разделе IDENTIFICATION DIVISION (РАЗДЕЛ ИДЕНТИФИКАЦИИ) обязательным является один параграф — PROGRAM-ID (ПРОГРАММА). В качестве имени программы употребляется слово КОБОЛа, определяемое программистом. Причем желательно, чтобы оно состояло не более чем из семи символов. В разделе ENVIRONMENT DIVISION (РАЗДЕЛ ОБОРУДОВАНИЯ) обязательной является секция CONFIGURATION SECTION (СЕКЦИЯ КОНФИГУРАЦИИ), состоящая из двух параграфов — SOURCE-COMPUTER (ИСХОДНАЯ-МАШИНА) и OBJECT-COMPUTER (РАБОЧАЯ-МАШИНА). Имена машин зафиксированы в компиляторе КОБОЛа для используемой вычислительной машины. Эти статьи и еще одна статья из параграфа FILE-CONTROL (УПРАВЛЕНИЕ-ФАЙЛАМИ) зависят от конкретной установки. Раздел ENVIRONMENT DIVISION помогает обеспечить независимость языка КОБОЛ от его реализаций. Секция INPUT-OUTPUT (ВВОДА-ВЫВОДА) необходима для связи физических устройств вычислительной машины с их именами в КОБОЛ-программе, используемыми в разделах данных и процедур. Параграф FILE-CONTROL является единственным, который в данном примере содержит секция INPUT-OUTPUT. Однако в общем случае эта секция может содержать и несколько других параграфов, которые будут описаны в следующей главе. Параграф FILE-CONTROL имеет статью SELECT (ДЛЯ) и фразу ASSIGN TO (НАЗНАЧИТЬ), которые связывают программное имя-файла с именем-устройства вычислительной машины. Для организации взаимодействия программы с оборудованием машины, авторы компилятора должны определить допустимые имена устройств. Программист должен обладать некоторыми знаниями языка управления заданиями для того, чтобы управлять процессом компиляции исходной КОБОЛ-программы в объектную программу на машинном языке. Также должны быть написаны инструкции для оператора по установке катушек магнитных лент и съемных магнитных дисков.

```

IDENTIFICATION DIVISION.
PROGRAM-ID. имя-программы.

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.
SOURCE-COMPUTER. имя-машины.
OBJECT-COMPUTER. имя-машины,

INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT имя-файла ASSIGN TO имя-устройства.

DATA DIVISION.

FILE SECTION.

FD имя-файла
    LABEL RECORDS ARE STANDARD.
01 имя-записи.
    05 имя-данного-1 PICTURE IS строка-литер-1.
    05 имя-данного-2 PICTURE IS строка-литер-2.

WORKING-STORAGE SECTION.
01 имя-записи VALUE IS литерал PICTURE IS строка-литер.

PROCEDURE DIVISION.

имя-секции SECTION.

имя-параграфа.
    OPEN INPUT список-имен-файлов OUTPUT список-имен-файлов.
    READ имя-файла RECORD AT END повелительные-операторы.
    WRITE имя-записи FROM имя-записи.
    MOVE литерал TO имя-данного.
    CLOSE список-имен-файлов.
    ADD имя-данного-1 имя-данного-2 TO имя-данного-3 ROUNDED.
    SUBTRACT имя-данного-4 FROM имя-данного-2 GIVING
        имя-данного-3 ON SIZE ERROR повелительные-операторы.
    COMPUTE имя-данного = арифметическое-выражение.
    GO TO имя-процедуры.
    IF условие повелительные-операторы.
    STOP RUN.

```

Рис. 5.1. Ядро КОБОЛа.

Статья SELECT должна быть описана для каждого файла, упомянутого в КОБОЛ-программе.

Раздел DATA DIVISION (РАЗДЕЛ ДАННЫХ) состоит из двух секций — FILE-SECTION (СЕКЦИЯ ФАЙЛОВ) и WORKING-STORAGE SECTION (СЕКЦИЯ РАБОЧЕЙ ПАМЯТИ). Секция FILE-SECTION определяет состав области внутренней памяти вычислительной машины, используемой для хранения логических за-

писей, считываемых из файла или записываемых в файл. Статья-описания-файла состоит из статьи индикатора-уровня-файла, за которой следует статья-описания-записи. Статья индикатора-уровня-файла начинается с зарезервированного слова FD (ОФ), за которым следует имя-файла (которое было задано в параграфе FILE-CONTROL) и фраза LABEL RECORD IS STANDARD (МЕТКИ СТАНДАРТНЫ) или OMITTED (ОПУЩЕНЫ). Статья-описания-записи начинается с индикатора-уровня 01, за которым следует имя-записи. Если в записи нет больше подразделений, то эта статья завершается фразой PICTURE (ШАБЛОН). Если в иерархии описания-записи есть групповые или элементарные данные, то им соответствуют свои индикаторы-уровней, имеющие собственные имена-данных. Фраза PICTURE может появиться только на уровне элементарного данного и специфицирует тип и размер этого данного. Статья-описания-данных в секции WORKING-STORAGE-SECTION имеет ту же структуру, какую имеют статьи-описания-данных в секции FILE-SECTION, за исключением того, что в ней разрешается использование фразы VALUE (ЗНАЧЕНИЕ).

Секция WORKING-STORAGE-SECTION резервирует область памяти для программно-генерируемых данных. Эти данные могут иметь начальные значения, присваиваемые при загрузке объектной программы. Эта область памяти не меняется при выполнении операторов READ (ЧИТАТЬ) или WRITE (ПИСАТЬ) в отличие от области памяти, определенной в секции FILE-SECTION.

Заголовки разделов и имена параграфов начинаются в поле А формата представления стандартного КОБОЛа, в то время как статьи, операторы и индикаторы-уровней, большие 01, начинаются в поле В.

Операторы раздела процедур сгруппированы в предложения, завершающиеся точкой, после которой следует по крайней мере один пробел. Отдельные операторы часто составляют одно предложение. Предложения могут быть объединены в параграфы. Параграфы могут быть сгруппированы в секции. Если какие-либо параграфы объединены в секцию, то и все остальные параграфы должны также быть включены в секции. На рис. 5.1 приведены операторы, задающие действия. Все файлы должны быть открыты до начала чтения или записи и должны быть закрыты, если в дальнейшем они не понадобятся. Оператор MOVE (ПОМЕСТИТЬ) осуществляет редактирование и передачу данных. Поэтому необходимо внимательно употреблять фразу PICTURE в посылающих и принимающих полях. Арифметические операторы допускают использование только элементарных цифровых данных, хотя фраза GIVING (ПОЛУЧАЯ) позволяет получать представление окончательного результата в цифровом-редактированном виде.

Для иллюстрации комбинированного использования этих возможностей КОБОЛа в этой главе приводятся пять программ с ком-

ментариями. Каждая программа написана самостоятельно. Но все пять программ связаны между собой, и одна из них содержит обращение к одной из других программ. Связующим звеном этих программ является гипотетический файл сотрудников, составленный из сокращенных записей, содержащих информацию о служащих.

Оператор **DISPLAY** (ВЫДАТЬ) весьма полезен при отладке программ и дает возможность проверить правильность промежуточных результатов. Для единственного другого оператора вывода, оператора **WRITE**, требуется использование статьи-описания-файла, в которой указывается спецификация формата записи и обозначение физического устройства. Оператор **DISPLAY** не требует описаний такого рода и намного проще в использовании. Кроме того, после отладки программы, операторы **DISPLAY** могут быть удалены из раздела процедур. После этого исходная программа может быть откомпилирована с целью получения окончательной рабочей программы. Формальное описание оператора **DISPLAY** имеет вид

$$\underline{\text{DISPLAY}} \left\{ \begin{array}{l} \text{литерал-1} \\ \text{имя-данного-1} \end{array} \right\} \dots$$

Например:

DISPLAY ANSWER-X.

DISPLAY "RESULTS OF FIRST CALCULATION ARE"
ANSWER-X RESULT-Y.

DISPLAY "END OF LOOP".

Значения данных, определяемые именем-данного-1 и т. д., преобразуются к виду, допускаемому устройствами вывода вычислительной машины, при этом от программиста не требуется управления форматами. Если длина выдаваемого данного, определяемого каким-либо именем-данного, превышает размеры одной записи или одной печатающей строки (чаще всего выдача осуществляется на печатающее устройство), то остаток данного переносится на следующую строку. Примеры употребления операторов **DISPLAY** показаны в приводимых программах, где они используются для выдачи результатов промежуточных вычислений с тем, чтобы контролировать правильность работы программы в моменты прохождения управления через параграфы.

5.2. Перепись файла карт в файл изменений

Информацию, поступающую с перфокарт, не следует передавать сразу же в основную программу. Предпочтительнее эту информацию поместить во вспомогательный файл и с помощью специальной программы подвергнуть данные контролю и выявить содержащиеся в них ошибки. В системе обработки данных не всегда возможно пре-

дотворить поступление неверной информации. Поэтому необходимо использовать всякую возможность обнаружения и исправления ошибок. В целях предотвращения ошибок при перфорации данных каждый документ обычно перфорируется дважды: сначала на перфораторе, а затем на устройстве, называемом верификатором (контроллером). Это обеспечивает контроль информации, так как на верификаторе производится сравнение набиваемых карт с картами, полученными на перфораторе. При передаче карт часто подсчитывается их количество для гарантии того, что ни одна из них не потерялась в процессе передачи. Обычно программа обнаружения неверных данных является самостоятельным шагом в схеме функционирования системы обработки данных и часто называется программой обеспечения достоверности. Реакция на ошибки представляет собой самостоятельную проблему. В нашем примере она состоит в выдаче сообщения об ошибке и ошибочной записи на печать для уведомления, последующей коррекции и считывания очередной карты из файла. Записи только из тех карт, которые прошли проверку, заносятся в файл изменений (TRANSACTION-FILE).

Таким образом, основное назначение рассматриваемой программы состоит в чтении карт, проверке правильности содержащихся в них данных и записи правильных образцов карт в файл изменений, который служит входным файлом для других программ. Кроме того, в результате выполнения программы, должен быть изготовлен список обнаруженных в картах ошибок для того, чтобы пользователь мог внести исправления.

На рис. 5.2 изображена блок-схема программы с именем PROG 520. Блок-схема дает общее представление о тех шагах, которые должны быть заданы в разделе процедур этой программы. Прежде всего выполняется ряд начальных шагов: устанавливаются начальные значения двух переменных и открываются используемые файлы. Затем осуществляется вход в основной цикл программы, начинающийся с ввода очередной карты с данными. Если входной файл карт пуст, то производится выход из основного цикла и работа программы заканчивается. Выход и остановка должны также произойти после того, как будут считаны все карты. Самым важным информационным полем в карте является идентификационный номер сотрудника, состоящий из десяти цифр и однозначно определяющий сотрудника. Для этого номера прежде всего осуществляется проверка, является ли он числовым, и если нет, то выдается сообщение об ошибке. Ошибка будет обнаружена в том случае, если в колонках, предназначенных для размещения идентификационного номера (ID-номера¹⁾), будут отперфорированы какие-либо алфавитные или

¹⁾ В дальнейшем вместо слов «идентификационный номер» будем употреблять сокращение ID-номер, где ID — сокращение от английского слова IDENTIFICATION — *Прим. перев.*

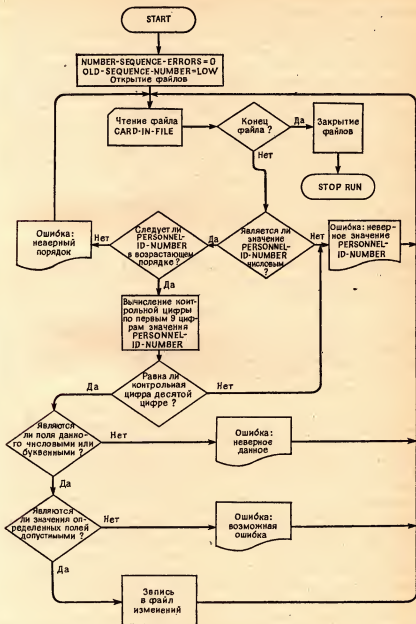


Рис. 5.2. Блок-схема переписи информации с карт на ленту.

специальные символы. Если ID-номер состоит из десяти цифр (любые пробелы также приведут к выходу по ошибке), то производятся действия по проверке правильности следования карт. Для удобства обработки файла сотрудников в последующих программах, файл изменений необходимо формировать в порядке возрастания ID-номеров. Это может быть достигнуто путем механической сортировки карт до ввода их этой программой. При этом необходимо проверять при чтении правильность следования карт во входном файле.

Если очередная карта имеет значение ID-номера, большее, чем у предыдущей карты, то проверяется правильность написания (перфорации) значения ID-номера. Один из методов такой проверки заключается в выполнении процедуры *вычисления контрольной цифры*. Контрольная цифра является одной из цифр ID-номера и вычисляется по определенному правилу по значениям остальных цифр. Например, в трехзначном числе

415

последняя цифра может быть вычислена как сумма первых двух цифр ($4+1=5$). Если бы в этом числе при передаче изменилась одна цифра, например оно стало бы

615

то нарушилось бы соответствие между контрольной цифрой и остальными цифрами числа, вычисляемое по правилу «сумма первых двух», что привело бы к обнаружению ошибки. Сумма цифр должна всегда давать одну цифру, поэтому из всей суммы оставляется только одна позиция, например

774

Это правило является довольно простым, но не позволяет обнаруживать все ошибки. Если в числе 415 первые две цифры поменяются местами, то в неверном числе 145 ошибка не будет обнаружена. Такая же ситуация возникает, если одновременно произойдут две компенсирующих друг друга ошибки, результатом которых будет, например, число 325. Фактически не существует правила вычисления контрольной цифры, способного обнаруживать все возможные ошибки. Однако тщательно выбранный метод вычисления контрольной цифры позволяет существенно понизить вероятность необнаружения ошибки при передаче чисел и обеспечить высокую надежность контроля правильности. В предлагаемом здесь методе вычисления контрольной цифры десятая цифра идентификационного номера сотрудника вычисляется по следующему правилу:

- 1) вычисляется сумма из первой, четвертой и седьмой цифр;
- 2) вычисляется сумма из второй, пятой и восьмой цифр, результат умножается на три;

- 3) вычисляется сумма из третьей, шестой и девятой цифр, результат умножается на девять;
- 4) подсчитывается сумма трех предварительных результатов, из которой в качестве контрольной цифры выбирается последняя цифра.

С помощью этой процедуры выявляются все единичные ошибки в цифрах чисел, все ошибки, связанные с перестановками цифр в числе, и большинство множественных ошибок в числе. Выполнение указанных вычислений для человека затруднительно, но для машины это не составит труда и не отнимет много времени. Приведем пример вычисления контрольной цифры для числа

1231231232

$1 + 1 + 1 = 3$

$(2 + 2 + 2) \times 3 = 18$

$(3 + 3 + 3) \times 9 = 81$

Общая сумма: 102

Контрольная цифра: 2

Если соответствующая карточная запись успешно проходит эту проверку, то производится следующее испытание, которое заключается в том, что проверяется, являются ли отдельные области данных целиком числовыми или целиком буквенными. Данные, которые могут представлять произвольную комбинацию символов, такие, как название и номер улицы, не могут подвергаться такой проверке.

Другой возможной проверкой называется проверка на допустимость. Значения некоторых данных в соответствии с их смыслом должны находиться в определенных пределах. Например, размеры оклада должны быть в пределах от 3.000 до 28.000 долларов в год. Если значение данного выходит за пределы, то оно может быть и верным, но оно должно быть проверено пользователем перед тем, как будет обрабатываться дальше. Этот метод контроля не допускает печати чеков на миллион долларов и представляет собой еще одно решето, через которое просеиваются данные.

Сама программа имеет вид:

IDENTIFICATION DIVISION.

PROGRAM-ID. PROG520.

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER. COMPUTER-NAME.

OBJECT-COMPUTER. COMPUTER-NAME.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

SELECT CARD-IN-FILE	ASSIGN TO READER.
SELECT TRANSACTION-FILE	ASSIGN TO TAPE.
SELECT ERROR-MESSAGE-FILE	ASSIGN TO PRINTER.

DATA DIVISION.

FILE SECTION.

FD CARD-IN-FILE

LABEL RECORD IS STANDARD.

01 INPUT-RECORD.

05 PERSONNEL-ID-NUMBER	PICTURE IS X(10).
05 PERSONNEL-NAME	PICTURE IS X(25).
05 PERS-SEX	PICTURE IS X.
05 PERS-AGE	PICTURE IS X(2).
05 PERS-HEIGHT	PICTURE IS X(3).
05 PERS-JOB-SKILL	PICTURE IS X(8).
05 PERS-ANNUAL-SALARY	PICTURE IS X(7).
05 PERS-DATE-LAST-PROMOTED	PICTURE IS X(6).
05 PERS-WEIGHT	PICTURE IS X(3).
05 PERS-HOME-STATE	PICTURE IS X(2).
05 PERS-DEPENDENTS	PICTURE IS X(2).
05 PERS-MERIT-RATING	PICTURE IS X(3).
05 PERS-SENIORITY-RATING	PICTURE IS X(2).
05 PERS-DATE-HIRED	PICTURE IS X(6).

FD TRANSACTION-FILE

LABEL RECORD IS STANDARD.

01 TRANSACTION-RECORD	PICTURE IS X(80).
-----------------------	-------------------

FD ERROR-MESSAGE-FILE

LABEL RECORD IS STANDARD.

01 ERROR-MESSAGE-RECORD	PICTURE IS X(132).
-------------------------	--------------------

WORKING-STORAGE SECTION.

01 ID-TEST-AREA.

05 DIGIT-1	PICTURE IS 9.
05 DIGIT-2	PICTURE IS 9.
05 DIGIT-3	PICTURE IS 9.
05 DIGIT-4	PICTURE IS 9.
05 DIGIT-5	PICTURE IS 9.
05 DIGIT-6	PICTURE IS 9.
05 DIGIT-7	PICTURE IS 9.
05 DIGIT-8	PICTURE IS 9.
05 DIGIT-9	PICTURE IS 9.
05 DIGIT-10	PICTURE IS 9.
01 NUMBER-OF-SEQUENCE-ERRORS	PICTURE IS 9(2).
01 OLD-SEQUENCE-NUMBER	PICTURE IS 9(10).
01 NEW-SEQUENCE-NUMBER	PICTURE IS 9(10).
01 TEST-DIGIT	PICTURE IS 9.
01 ERROR-ONE PICTURE IS X(26)	VALUE IS
"INPUT CARD OUT OF SEQUENCE".	
01 ERROR-TWO PICTURE IS X(48)	VALUE IS
"MORE THAN ONE CARD OUT OF	
SEQUENCE—RUN STOPPED".	
01 ERROR-THREE PICTURE IS X(35)	VALUE IS
"INVALID PERSONNEL NUMBER	
DISCOVERED".	
01 ERROR-FOUR PICTURE IS X(23)	VALUE IS
"A DATA ITEM IS IN ERROR".	
01 ERROR-FIVE PICTURE IS X(43)	VALUE IS
"PROBABLE ERROR SINCE A DATA ITEM	
IS UNUSUAL".	

PROCEDURE DIVISION.

INITIALIZE-PARAGRAPH.

MOVE ZERO TO NUMBER-OF-SEQUENCE-ERRORS
OLD-SEQUENCE-NUMBER.

OPEN INPUT CARD-IN-FILE OUTPUT
TRANSACTION-FILE ERROR-MESSAGE-FILE.

READ-DATA-PARAGRAPH.

READ CARD-IN-FILE RECORD AT END

CLOSE CARD-IN-TRANSACTION-FILE

ERROR-MESSAGE-FILE STOP RUN.

IF PERSONNEL-ID-NUMBER IS NOT NUMERIC

GO TO PERSONNEL-ID-ERROR.

VALIDATE-ID-NUMBER.

MOVE PERSONNEL-ID-NUMBER TO ID-TEST-AREA.

COMPUTE TEST-DIGIT = DIGIT-1 + DIGIT-4 +

DIGIT-7 + 3 * (DIGIT-2 + DIGIT-5 + DIGIT-8) +

9 * (DIGIT-3 + DIGIT-6 + DIGIT-9).

IF TEST-DIGIT IS EQUAL TO DIGIT-10 GO TO

VALIDATE-DATA-ITEMS.

DISPLAY TEST-DIGIT.

CHECK-SEQUENCE-ORDER.

DISPLAY PERSONNEL-ID-NUMBER OLD-SEQUENCE-
NUMBER.

MOVE PERSONNEL-ID-NUMBER TO NEW-SEQUENCE-
NUMBER.

IF NEW-SEQUENCE-NUMBER IS GREATER THAN OLD-
SEQUENCE-NUMBER MOVE NEW-SEQUENCE-

NUMBER TO OLD-SEQUENCE-NUMBER

GO TO VALIDATE-ID NUMBER.

WRITE ERROR-MESSAGE-RECORD FROM ERROR-ONE.

WRITE ERROR-MESSAGE-RECORD FROM INPUT-
RECORD.

ADD 1 TO NUMBER-OF-SEQUENCE-ERRORS.

IF NUMBER-OF-SEQUENCE-ERRORS IS LESS THAN 2
GO TO READ-DATA-PARAGRAPH.

WRITE ERROR-MESSAGE-RECORD FROM ERROR-TWO.
STOP RUN.

PERSONNEL-ID-ERROR.

WRITE ERROR-MESSAGE-RECORD FROM ERROR-THREE.

WRITE-CARD-IMAGE.

WRITE ERROR-MESSAGE-RECORD FROM INPUT-RECORD.

GO TO READ-DATA-PARAGRAPH.

VALIDATE-DATA-ITEMS.

IF PERS-JOB-SKILL NOT NUMERIC

OR PERS-DATE-LAST-PROMOTED NOT NUMERIC

OR PERS-MERIT-RATING NOT NUMERIC

OR PERS-DATE-HIRED NOT NUMERIC

OR PERS-SEX NOT ALPHABETIC

OR PERS-HOME-STATE NOT ALPHABETIC GO TO DATA-ITEMS-ERROR.

IF PERS-AGE IS GREATER THAN 70 GO TO PROBABLE-ERROR.

IF PERS-HEIGHT IS NOT EQUAL TO ZERO AND

(PERS-HEIGHT IS LESS THAN 40 OR

PERS-HEIGHT IS GREATER THAN 85)

GO TO PROBABLE-ERROR.

IF PERS-DEPENDENTS IS GREATER THAN 20 GO TO PROBABLE-ERROR.

IF PERS-ANNUAL-SALARY IS GREATER THAN 3000

OR PERS-ANNUAL-SALARY IS LESS THAN 2800

GO TO WRITE-TRANSACTION-FILE ELSE GO TO

PROBABLE-ERROR.

DATA-ITEMS-ERROR.

WRITE ERROR-MESSAGE-RECORD FROM ERROR-FOUR.

GO TO WRITE-CARD-IMAGE.

PROBABLE-ERROR.

WRITE ERROR-MESSAGE-RECORD FROM ERROR-FIVE.

GO TO WRITE-CARD-IMAGE.

WRITE-TRANSACTION-FILE.

WRITE TRANSACTION-RECORD FROM INPUT-RECORD.
GO TO READ-DATA-PARAGRAPH.

Файл CARD-IN-FILE является набором записей, именуемых INPUT-RECORD, которые хранятся в колоде перфокарт, предназначенной для устройства ввода, именуемого READER. Эти карты были автономно отперфорированы на перфораторах вычислительного центра в соответствии с форматом, представленным в секции файлов раздела данных. Заметим, что каждое поле в записи INPUT-RECORD обычно описывается фразой PICTURE как буквенно-цифровое (ALPHABETIC) данное, хотя фактический класс данного может быть числовым. Это вызвано тем, что класс ALPHABETIC в формате обеспечивает ввод символов с перфокарт в том виде, как они были отперфорированы. Фактический же класс данного определяется после ввода карты в машину по правилу, изложенному выше.

В случае употребления для числовых данных формата NUMERIC в данном поле могут находиться только цифры, и этот факт обычно учитывает вычислительная машина (точнее программа ввода). Так в этом случае обычно при чтении принимаются во внимание только числовые позиции в колонках карт, т. е. буква A будет воспринята как 1, буква B — как 2 и т. п. Последующая проверка условия на принадлежность класса данного к числовому классу ошибочно показала бы, что данное является числовым. Только описание входного данного как буквенно-цифрового обеспечит ввод любого символа во внутреннюю память вычислительной машины в точном соответствии с его изображением на карте (буква A, отперфорированная на карте, будет считана как A).

В разделе PROCEDURE DIVISION после выполнения начальных действий будет считана одна карточная запись и ее ID-номер будет проверен на принадлежность к числовому классу. Затем это значение сравнивается со значением данного OLD-SEQUENCE-NUMBER для того, чтобы убедиться, что карточные записи поступают в машину в возрастающем порядке содержащихся в них ID-номеров. Эта проверка дает гарантию того, что файл TRANSACTION-FILE будет представлять упорядоченную по значениям ID-номера последовательность, что позволит облегчить поиск определенной записи в последующих программах. В последовательности обычно допускается лишь одна запись, не удовлетворяющая критерию упорядоченности. Появление второй такой записи приводит к прекращению работы программы. В целях упрощения эта ошибочная ситуация не показана на блок-схеме. Однако такой подход, когда при накоплении фиксированного числа ошибок работа останавливается, является типичным. Первоначально эти вводимые карты упорядочиваются путем сортировки на специальном устройстве независимо от вычислительной машины. Различные сообщения об ошиб-

ках переносятся из областей рабочей-памяти в область файла для вывода. При этом сообщения об ошибках должны передаваться в область файла выдачи каждый раз, когда они необходимы, так как каждый оператор WRITE разрушает выводимую запись в области выходного файла.

В результате выполнения этой программы и считывания колоды перфокарт будет сформирован проконтролированный файл изменений, являющийся выходным для данной программы и входным для следующей программы, описанной ниже. Кроме того, программа формирует список ошибок, который в системе обработки данных необходимо передать службе управления обработкой, где ошибки будут исправлены, и скорректированные данные затем будут отправлены на перфорацию для повторного ввода при очередном выполнении программы.

Упражнение

Преобразовать файл на картах в последовательный файл на ленте. Напишите исходную КОБОЛ-программу, предварительно нарисовав ее блок-схему. Предполагается, что файл перфокарт уже существует и упорядочен по возрастанию идентификационных номеров товара. Каждая карточная запись, состоящая из 80 колонок, содержит идентификационный номер товара, состоящий из четырех цифр, буквенное наименование товара, состоящее не более чем из двадцати пяти букв, числовой код из семи цифр, определяющий поставщика товара, и числовое поле из шести цифр, определяющее количество единиц данного товара на текущий момент. Файл на картах упорядочен только по идентификационному номеру товара, а не по коду поставщика. Одновременно с записью карт на магнитную ленту программа должна распечатывать карты для обеспечения визуального контроля. Кроме того, выполните все возможные проверки данных и печатайте сообщение об ошибке сразу же после обработки ошибочной карты.

5.3. Обновление файла сотрудников

Файл сотрудников (OLD-PERSONNEL-FILE) представляет собой набор записей, формат которых был описан в предыдущем разделе. Каждая запись содержит сведения об отдельном сотруднике. В этих записях описывается лишь небольшая доля той информации, которую можно было бы в них хранить. Файл сотрудников должен корректироваться при появлении новых сотрудников или при уходе старых, или при изменении значений каких-то данных. В результате корректировки могут меняться отдельные поля, а иногда и целые записи. Для корректировки такого файла вводятся три операции:

- а) добавление записей,
- б) удаление записей,
- в) изменение значений данных.

В примере программы, приводимой в данном разделе, файл изменений из предыдущего раздела считывается и используется в качестве исходной информации для изменения файла сотрудников. Расплачиваясь за ограничения, свойственные последовательным файлам и описанные в гл. 2, процедура обработки вынуждена считывать старые записи файла сотрудников и переписывать обработанные записи в новый файл. Такой подход приводит к копированию всего файла сотрудников. Даже в том случае, когда требуется изменить только одну запись, файл сотрудников должен быть целиком считан и скопирован на новый файл. По этой причине целесообразно *накапливать* достаточное количество записей в файле изменений, чтобы работа вычислительной машины была оправдана.

Файл изменений и файл сотрудников упорядочены по одному и тому же параметру — идентификационному номеру сотрудника. Это делает возможным создание новой копии и выполнение процесса внесения изменений. ID-номер используется либо для установления соответствия записей из двух файлов, либо для идентификации новой записи, порождаемой по очередной записи из файла изменений, аналога которой не оказалось в файле сотрудников. Пусть перед началом обработки файл сотрудников и файл изменений имеют следующие последовательности ID-номеров:

Файл изменений	Файл сотрудников
0000123451	0000100003
0000200006	0000150008
0000250001	0000200006
0000395005	0000250001
	0000300009
	0000350004
	0000400002
	0000450007

Первой записи из файла изменений не соответствует ни одна запись в файле сотрудников. В этом случае происходит добавление новой записи, соответствующей новому сотруднику. ID-номера следующих двух записей файла изменений имеются в обоих файлах. В этом случае могут быть либо удаления, либо изменения записей файла сотрудников. Последняя запись в файле изменений с ID-номером, равным 0000395995, порождает еще одну новую запись в файле сотрудников. После обработки обновленный файл сотрудников (NEW-PERSONNEL-FILE) будет выглядеть следующим образом (учитывая, что запись с номером 0000200006 была удалена):

Новый файл сотрудников

0000100003
0000123451
0000150008
0000250001 (изменена)
0000300009
0000350004
0000395005
0000400002
0000450007

Для того чтобы очередная порция изменений могла быть внесена в этот новый файл, он должен быть упорядочен так же, как и старый, т. е. по возрастанию ID-номеров. Новый файл является физически новым файлом, потому что ограничения последовательного доступа приводят к невозможности перезаписи на входной (INPUT) файл. Старый файл сотрудников перестает существовать, а новый файл сотрудников становится старым файлом сотрудников для очередного цикла или для другого вида обработки. Слежение за процессом этих изменений, выполнение определенных действий по удалению старого файла и модификация меток может быть довольно трудоемкой работой на больших установках. Идентификационные номера сотрудников имеют весьма важное значение для этого процесса. Поэтому так важно, чтобы эти номера записывались с контрольной цифрой во избежание удаления записи не того сотрудника.

Другая проблема заключается в способе, с помощью которого старый файл сотрудников формируется в первый раз. Это несколько похоже на поиски начала кольца, так как старый файл преобразуется в новый файл, который должен быть сразу же назван старым файлом с тем, чтобы служить входным файлом для очередного процесса внесения изменений. В примере программы, приводимой в этом разделе, первоначально имеется пустой, но помеченный старый файл сотрудников, из которого с помощью добавления к нему записей изменений создается новый файл сотрудников. При последующем выполнении программы уже возможны удаления, изменения, а также и добавления записей. Процесс присваивания меток управляется диспетчером базы данных и не рассматривается в данном примере.

Блок-схема, представленная на рис. 5.3, изображает принцип обработки, основанный на сравнении ID-номеров сотрудников из файла изменений с ID-номерами сотрудников из старого файла. Поскольку исчерпывание любого из файлов не означает, что выполнение программы прекращается, каждая фраза AT END помещает значение 9999999999 в поле соответствующего ID-номера, обеспечивая тем самым несовпадение номеров в последующих сравнениях.

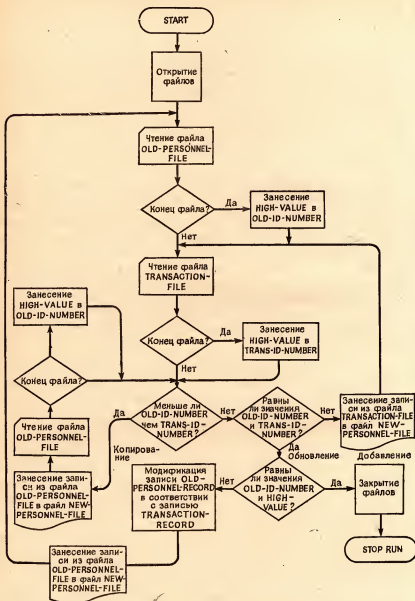


Рис. 5.3. Блок-схема внесения изменений.

После этого совпадение номеров возможно только в том случае, когда в полях ID-номеров обеих записей будут стоять одни девятки. При этом программа прекратит работу. Вместо девяток нельзя использовать стандартную константу HIGH-VALUE (НАИБОЛЬШЕЕ-ЗНАЧЕНИЕ), так как она имеет нечисловое значение, которое не может быть помещено в числовое поле.

Если проследовать по блок-схеме, используя ранее приведенный пример файла, то в результате выполнения первого оператора READ старый ID-номер (OLD-ID-NUMBER) будет равен 0000100003. После выполнения второго оператора READ ID-номер изменения (TRANS-ID-NUMBER) становится равным 0000123451. При сравнении обнаруживается, что старый ID-номер меньше, чем ID-номер изменения, и осуществляется переход к процессу копирования (ветвь COPY), который заключается в переписывании записи из старого-файла-сотрудников в новый-файл-сотрудников. Затем считывается следующая запись из старого файла, и старый ID-номер становится равным 0000150008. На этот раз при сравнении двух ID-номеров старый ID-номер оказывается не меньше ID-номера изменений (который сохраняет прежнее значение 0000123451) и не равен ему. В этом случае осуществляется переход к процессу добавления новой записи к новому-файлу-сотрудников (ветвь ADD). При этом в новый файл сотрудников записывается запись-изменение (TRANSACTION-RECORD). После этого считывается очередная запись файла изменений, и ID-номер изменения становится равным 0000200006. Это приводит к переходу на ветвь COPY и считыванию очередной записи из старого-файла, которая в точности согласуется с записью-изменения по ID-номеру, в результате чего следует переход к процессу обновления (ветвь UPDATE). После изменения или удаления записи из старого-файла-сотрудников управление программой возвращается в начало схемы для продолжения процесса обработки.

В нашем примере первым исчерпается файл-изменений, и в ID-номер изменения будет записано значение 9999999999. Ветвь COPY копирует записи из старого-файла-сотрудников до его исчерпывания, и в конце концов осуществляется выход к операторам CLOSE и STOP RUN. Программа, реализующая алгоритм, представленный блок-схемой, имеет вид

IDENTIFICATION DIVISION.

PROGRAM-ID. PROG530.

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER. COMPUTER-NAME.

OBJECT-COMPUTER. COMPUTER-NAME.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

SELECT TRANSACTION-FILE	ASSIGN TO TAPE.
SELECT OLD-PERSONNEL-FILE	ASSIGN TO TAPE.
SELECT NEW-PERSONNEL-FILE	ASSIGN TO TAPE.

DATA DIVISION.

FILE SECTION.

FD TRANSACTION-FILE

LABEL RECORDS ARE STANDARD.

01 TRANSACTION-RECORD.

05 TRANS-ID-NUMBER	PICTURE IS 9(10).
05 TRANS-NAME	PICTURE IS X(25).
05 TRANS-SEX	PICTURE IS A.
05 TRANS-AGE	PICTURE IS 99.
05 TRANS-HEIGHT	PICTURE IS 99V9.
05 TRANS-JOB-SKILL	PICTURE IS 9(8).
05 TRANS-ANNUAL-SALARY	PICTURE IS 9(5)V9(2).
05 TRANS-DATE-LAST- PROMOTED	PICTURE IS 9(6).
05 TRANS-WEIGHT	PICTURE IS 9(3).
05 TRANS-HOME-STATE	PICTURE IS A(2).
05 TRANS-DEPENDENTS	PICTURE IS 9(2).
05 TRANS-MERIT-RATING	PICTURE IS 9(3).
05 TRANS-SENIORITY- RATING	PICTURE IS 9(2).
05 TRANS-DATE-HIRED	PICTURE IS 9(6)..

FD OLD-PERSONNEL-FILE

LABEL RECORDS ARE STANDARD.

01 OLD-PERSONNEL-RECORD.

05 OLD-ID-NUMBER	PICTURE IS 9(10).
05 OLD-NAME	PICTURE IS X(25).
05 OLD-SEX	PICTURE IS A.
05 OLD-AGE	PICTURE IS 99.
05 OLD-HEIGHT	PICTURE IS 99V9.
05 OLD-JOB-SKILL	PICTURE IS 9(8).

05 OLD-ANNUAL-SALARY	PICTURE IS 9(5)V9(2).
05 OLD-DATE-LAST- PROMOTED	PICTURE IS 9(6).
05 OLD-WEIGHT	PICTURE IS 9(3).
05 OLD-HOME-STATE	PICTURE IS A(2).
05 OLD-DEPENDENTS	PICTURE IS 9(2).
05 OLD-MERIT-RATING	PICTURE IS 9(3).
05 OLD-SENIORITY- RATING	PICTURE IS 9(2).
05 OLD-DATE-HIRED	PICTURE IS 9(6).

FD NEW-PERSONNEL-FILE

LABEL RECORDS ARE STANDARD.

01 NEW-PERSONNEL-RECORD PICTURE IS X(80).

PROCEDURE DIVISION.

FIRST-PARAGRAPH.

OPEN INPUT TRANSACTION-FILE OLD-PERSONNEL-
FILE.

OPEN OUTPUT NEW-PERSONNEL-FILE.

READ-OLD-PERSONNEL-FILE.

READ OLD-PERSONNEL-FILE AT END
MOVE 9999999999 TO OLD-ID-NUMBER.

READ-TRANSACTION-FILE.

READ TRANSACTION-FILE AT END
MOVE 9999999999 TO TRANS-ID-NUMBER.

COMPARE-ID-NUMBERS.

DISPLAY OLD-ID-NUMBER TRANS-ID-NUMBER.
IF OLD-ID-NUMBER IS LESS THAN TRANS-ID-NUMBER
GO TO COPY-OLD-RECORD ELSE IF OLD-ID-
NUMBER IS EQUAL TO TRANS-ID-NUMBER
GO TO UPDATE-OLD-RECORD
ELSE GO TO ADD-NEW-RECORD.

COPY-OLD-RECORD.

WRITE NEW-PERSONNEL-RECORD FROM
OLD-PERSONNEL-RECORD.
READ OLD-PERSONNEL-FILE AT END
MOVE 9999999999 TO OLD-ID-NUMBER.
GO TO COMPARE-ID-NUMBERS.

UPDATE-OLD-RECORD.

IF OLD-ID-NUMBER IS EQUAL TO 9999999999
CLOSE TRANSACTION-FILE
OLD-PERSONNEL-FILE
NEW-PERSONNEL-FILE
STOP RUN.
IF TRANS-NAME IS EQUAL TO "DELETE"
GO TO READ-OLD-PERSONNEL-FILE.
IF TRANS-NAME IS NOT EQUAL TO ALL SPACES
MOVE TRANS-NAME TO OLD-NAME.
IF TRANS-SEX IS NOT EQUAL TO ALL SPACES
MOVE TRANS-SEX TO OLD-SEX.
IF TRANS-AGE IS NOT EQUAL TO ZERO
MOVE TRANS-AGE TO OLD-AGE.
IF TRANS-HEIGHT IS NOT EQUAL TO ZERO
MOVE TRANS-HEIGHT TO OLD-HEIGHT.
IF TRANS-JOB-SKILL IS NOT EQUAL TO ZERO
MOVE TRANS-JOB-SKILL TO OLD-JOB-SKILL.
IF TRANS-ANNUAL-SALARY IS NOT EQUAL TO ZERO
MOVE TRANS-ANNUAL-SALARY TO
OLD-ANNUAL-SALARY.
IF TRANS-DATE-LAST-PROMOTED IS NOT EQUAL
TO ZERO MOVE TRANS-DATE-LAST-PROMOTED
TO OLD-DATE-LAST-PROMOTED.
IF TRANS-WEIGHT IS NOT EQUAL TO ZERO
MOVE TRANS-WEIGHT TO OLD-WEIGHT.
IF TRANS-HOME-STATE IS NOT EQUAL TO ZERO
MOVE TRANS-NOME-STATE TO
OLD-HOME-STATE.
IF TRANS-DEPENDENTS IS NOT EQUAL TO ZERO
MOVE TRANS-DEPENDENTS TO

OLD-DEPENDENTS.

IF TRANS-DEPENDENTS IS EQUAL TO 99

MOVE ZERO TO OLD-DEPENDENTS.

IF TRANS-MERIT-RATING IS NOT EQUAL TO ZERO

MOVE TRANS-MERIT-RATING TO

OLD-MERIT-RATING.

IF TRANS-SENIORITY-RATING IS NOT EQUAL

TO ZERO MOVE TRANS-SENIORITY-RATING

TO OLD-SENIORITY-RATING.

IF TRANS-DATE-HIRED IS NOT EQUAL TO ZERO

MOVE TRANS-DATE-HIRED TO

OLD-DATE-HIRED.

WRITE NEW-PERSONNEL-RECORD FROM

OLD-PERSONNEL-RECORD.

GO TO READ-OLD-PERSONNEL-FILE.

ADD-NEW-RECORD.

DISPLAY TRANSACTION-RECORD.

WRITE NEW-PERSONNEL-RECORD FROM

TRANSACTION-RECORD.

GO TO READ-TRANSACTION-FILE.

Для всех трех файлов в программе в качестве носителя назначена магнитная лента (TAPE). Конкретное распределение устройств и установка требуемых катушек производится оператором через операционную систему. Групповое данное TRANSACTION-RECORD подразделяется на четырнадцать элементарных данных. Фраза PICTURE на этот раз определяет, к какому фактическому классу принадлежит данное: числовому, буквенному или буквенно-цифровому. В этой же фразе указывается положение десятичной точки. В предыдущей программе все входные данные квалифицировались как буквенно-цифровые с целью обеспечения возможности выполнения проверочных действий, в которых здесь нет необходимости. Запись OLD-PERSONNEL-RECORD по формату идентична записи TRANSACTION-RECORD. Заметим, что с целью облегчения чтения программы каждое элементарное данное именуется с учетом «родительской» записи при помощи общего префикса, приставляемого к каждому имени. Это не есть требование языка, но существенно помогает при чтении текста программы. Запись NEW-PERSONNEL-RECORD используется только для вывода, поэтому не нуждается в подразделении.

Раздел процедур достаточно подробно отражен на блок-схеме. Оператор IF (ЕСЛИ) в параграфе COMPARE представляет пример сложного условного оператора. При написании таких операторов необходимо позаботиться, чтобы варианты ELSE (ИНАЧЕ) были правильно согласованы. В данном случае оператор имеет следующую конструкцию:

IF условие повелительные-операторы
ELSE IF условие повелительные-операторы
ELSE повелительные-операторы

Параграф UPDATE-OLD-RECORD представляет собой конкретизацию действий, отраженных на блок-схеме (ветвь DELETE). Первоначально в параграфе проверяется поле TRANS-NAME, которое может содержать указание на удаление (DELETE) записи. Если таковое имеется, то происходит переход к параграфу READ-OLD-PERSONNEL-FILE, что фактически приводит к уничтожению текущей записи. В противном случае поля в записи OLD-PERSONNEL-RECORD заменяются соответствующими полями из записи TRANSACTION-RECORD, если последние непустые или ненулевые. Способ задания отсутствия изменений через пустое или нулевое значение поля вряд ли является наилучшим, в частности потому, что программа проверки достоверности из предыдущего раздела требует аккуратной перфорации карты с изменениями с тем, чтобы она соответствовала требованиям, предъявляемым к расположению информации на карте. Ввиду того что нулевое или пустое поле указывает на отсутствие в нем изменения, необходимо ввести некоторые особые средства для полей, задающих установку нулевого значения в качестве изменения. Например, для поля TRANS-DEPENDENTS строка 00 означает «изменений нет», в то время как строка 99 означает «установка в ноль».

Упражнение

Обновить файл инвентаризации. В первом упражнении был создан последовательный файл логических записей, содержащих идентификационный код товара, наименование товара, код поставщика и количество товара, имеющегося в наличии. Записи файла были упорядочены по идентификационному коду товара. В данном примере составьте программу чтения файла перфокарт, также упорядоченного по возрастанию идентификационного кода товара, каждая запись которого содержит код товара и поле числа со знаком, отражающее количество товара, добавляемого на склад (положительная величина), или количество товара, изымаемого со склада для продажи (отрицательная величина). Для каждой считанной карточной записи должна быть найдена соответствующая ей запись товара в файле на ленте и в ней изменено поле «количество

товара в наличии» путем алгебраического добавления к нему величины-изменения. Если в файле на ленте не отыщется запись, соответствующая входной записи, должно быть напечатано сообщение об ошибке на печатающем устройстве.

5.4. Обработка нарядов на работу

Предположим, что рассматривается пример программы обработки наряда (заказа) на работу, которая может быть выполнена одним из сотрудников, перечисленных в файле сотрудников, сформированном в предыдущих примерах. Первоначально предполагается, что среди данных о сотруднике имеется восьмизначный номер специальности (OLD-JOB-SKILL), по которому могут быть выбраны сотрудники, способные выполнить заданную работу. Просмотрим файл сотрудников и отыщем всех тех из них, которые имеют соответствующий номер специальности. Если таковых не окажется, то напечатаем об этом специальное сообщение. Если только один сотрудник имеет соответствующую специальность, тогда напечатаем его идентификационный номер и фамилию. Если несколько записей в файле содержит соответствующий номер специальности, то из этой группы выбираем того сотрудника, который обладает наивысшей квалификацией (MERIT-RATING). Если оказывается, что среди этих сотрудников имеется несколько с одинаковой квалификацией, то будем использовать стаж работы (SENIORITY-RATING) как окончательный критерий выбора. Если и в этом случае окажется больше одного кандидата, то выберем любого из них и напечатаем его ID-номер и фамилию.

В этом примере порядок следования записей в файле сотрудников может быть произвольно изменен по сравнению с предыдущими разделами. В целях упрощения рассматриваемой программы записи в файле сотрудников расположим по возрастанию номеров специальностей. Таким образом, поиск сотрудника с соответствующей специальностью заключается в последовательном просмотре файла до тех пор, пока либо не будет обнаружена соответствующая запись, либо номер специальности в файле сотрудников не станет превышать номер специальности, требуемой для выполнения работы. При такой организации файла сотрудников нет необходимости просматривать весь файл до конца, чтобы убедиться в отсутствии желаемой специальности.

Кроме того, в случае наличия нескольких записей с одинаковой специальностью они будут сгруппированы вместе. Пусть внутри этой группы записи будут упорядочены в соответствии с квалификацией и стажем работы, как упоминалось ранее. Изменение порядка следования записей в файле может быть выполнено программой сортировки, копирующей исходный файл в новый файл с требуемым

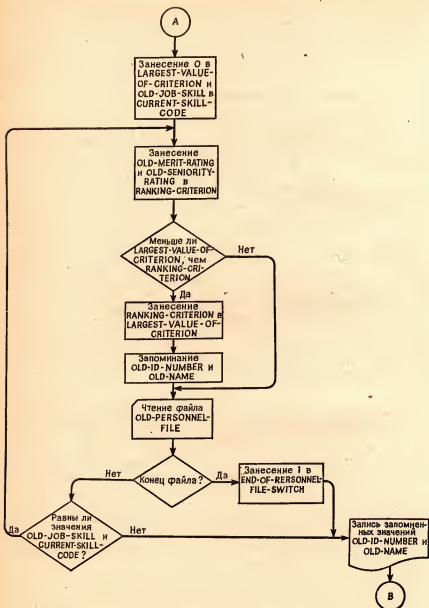


Рис. 5.4. Блок-схема обработки нарядов на работу.

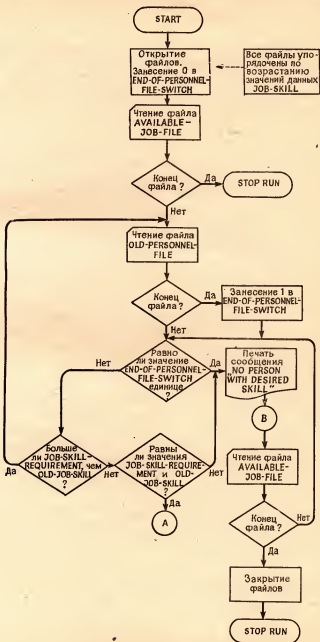


Рис. 5.4. (Продолжение)

порядком следования записей. Сортировка является операцией, которая часто встречается в коммерческих системах обработки данных.

Важно отметить, что порядок следования записей в файле полностью определяет процедуру решения задачи: «Есть ли в файле сотрудник с требуемой специальностью?» Невозможно писать программу, не зная структуры данных. *Структура данных* — это общий термин, под которым понимается организация символов в записях, записей в файлах и файлов на устройствах.

Именно поэтому при программировании такое большое внимание уделяется разделу ENVIRONMENT DIVISION, определяющему связи между логическими и физическими файлами, разделу DATA DIVISION, содержащему пунктуальное описание внутренней организации данных, выбору порядка следования записей в файле для обеспечения его последовательной обработки. Раздел DATA DIVISION, таким образом, фиксирует структуру данных для решаемой задачи. И поскольку эта обработка опирается на выбранную структуру, изменение структуры данных чаще всего приводит к тому, что программа перестает работать.

Блок-схема программы представлена на рис. 5.4. Первоначально открываются файлы и обнуляется переменная-переключатель (в программе она названа END-OF-PERSONNEL-FILE-SWITCH). Переключатель используется в программе только для фиксации момента исчерпания файла сотрудников. Такой метод часто используется в тех случаях, когда наступление события (в данном случае конец файла) может произойти в нескольких местах программы, но проверка (совершилось событие или нет) должна осуществляться только один раз.

Затем файл сотрудников считывается и по его исчерпыванию (AT END) в переключатель устанавливается значение 1. Это приводит к переходу на ветвь, по которой сначала печатается сообщение «нет соответствия» (NO MATCH), после чего (точка В в блок-схеме) считывается очередной наряд на работу. Точки А и В являются точками связи для блок-схемы. Такое изображение позволяет избежать части стрелок на блок-схеме и разбить ее на составные части, что делает ее лучше обозримой.

Программа обработки нарядов на работу имеет вид

IDENTIFICATION DIVISION.

PROGRAM-ID. PROG540.

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER. COMPUTER-NAME.

OBJECT-COMPUTER. COMPUTER-NAME.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

SELECT AVAILABLE-JOB-FILE	ASSIGN TO READER.
SELECT OLD-PERSONNEL-FILE	ASSIGN TO TAPE.
SELECT PRINT-FILE	ASSIGN TO PRINTER.

DATA DIVISION.

FILE SECTION.

FD AVAILABLE-JOB-FILE

LABEL RECORDS ARE STANDARD.

01 AVAILABLE-JOB-RECORD.

05 JOB-CODE-NUMBER	PICTURE IS X(9).
05 JOB-SKILL-REQUIREMENT	PICTURE IS X(8).
05 FILLER	PICTURE IS X(63).

FD OLD-PERSONNEL-FILE

LABEL RECORDS ARE STANDARD.

01 OLD-PERSONNEL-RECORD.

05 OLD-ID-NUMBER	PICTURE IS 9(10).
05 OLD-NAME	PICTURE IS X(25).
05 FILLER	PICTURE IS X(6).
05 OLD-JOB-SKILL	PICTURE IS 9(8).
05 FILLER	PICTURE IS X(20).
05 OLD-MERIT-RATING	PICTURE IS 9(3).
05 OLD-SENIORITY-RATING	PICTURE IS 9(2).
05 FILLER	PICTURE IS X(6).

FD PRINT-FILE

LABEL RECORDS ARE STANDARD.

01 PRINT-LINE.

05 PRINT-JOB-CODE	PICTURE IS X(9).
05 FILLER	PICTURE IS X(3).
05 PRINT-SKILL-REQUIREMENT	PICTURE IS X(8).
05 FILLER	PICTURE IS X(3).
05 PRINT-ID-NUMBER	PICTURE IS 9(10).
05 FILLER	PICTURE IS X(3).

05 PRINT-NAME	PICTURE IS X(25).
05 FILLER	PICTURE IS X(3).
05 PRINT-REMARKS	PICTURE IS X(45).

WORKING-STORAGE-SECTION.

01 END-OF-PERSONNEL-FILE-SWITCH	PICTURE IS 9.
01 LARGEST-VALUE-OF-CRITERION	PICTURE IS 9(5).
01 LARGEST-ID-NUMBER	PICTURE IS 9(10).
01 LARGEST-NAME	PICTURE IS X(25).
01 RANKING-CRITERION.	
05 RANKING-MERIT	PICTURE IS 9(3).
05 RANKING-SENIORITY	PICTURE IS 9(2).
01 CURRENT-SKILL-CODE	PICTURE IS 9(8).
01 NO-MATCH-MESSAGE	PICTURE IS X(28)
VALUE IS "NO PERSON WITH DESIRED SKILL".	

PROCEDURE DIVISION.

INITIAL-STEPS.

```

MOVE ZERO TO END-OF-PERSONNEL-FILE-SWITCH.
OPEN INPUT AVAILABLE-JOB-FILE
    OLD-PERSONNEL-FILE
    OUTPUT PRINT-FILE.
READ AVAILABLE-JOB-FILE RECORD AT END
CLOSE AVAILABLE-JOB-FILE
OLD-PERSONNEL-FILE PRINT-FILE STOP RUN.
```

READ-OLD-PERSONNEL-FILE.

```

READ OLD-PERSONNEL-FILE RECORD AT END
MOVE 1 TO END-OF-PERSONNEL-FILE-
SWITCH.
```

COMPARE-JOB-SKILLS.

```

IF END-OF-PERSONNEL-FILE-SWITCH IS EQUAL TO 1
GO TO NO-JOB-MATCH-FOUND.
IF JOB-SKILL-REQUIREMENT IS GREATER THAN
OLD-JOB-SKILL
```

GO TO READ-OLD-PERSONNEL-FILE.
IF JOB-SKILL-REQUIREMENT IS EQUAL TO
OLD-JOB-SKILL
GO TO FOUND-JOB-MATCH.

NO-JOB-MATCH-FOUND.
MOVE SPACES TO PRINT-LINE.
MOVE JOB-CODE-NUMBER TO PRINT-JOB-CODE.
MOVE NO-MATCH-MESSAGE TO PRINT-REMARKS.
WRITE PRINT-LINE.

READ-ANOTHER-JOB.
READ AVAILABLE-JOB-FILE RECORD AT END
CLOSE AVAILABLE-JOB-FILE
OLD-PERSONNEL-FILE PRINT-FILE STOP RUN.
GO TO COMPARE-JOB-SKILLS.

FOUND-JOB-MATCH.
MOVE ZERO TO LARGEST-VALUE-OF-CRITERION.
MOVE OLD-JOB-SKILL TO CURRENT-SKILL-CODE.

SELECT-BEST-LOOP.
MOVE OLD-MERIT-RATING TO RANKING-MERIT.
MOVE OLD-SENIORITY-RATING TO
RANKING-SENIORITY.
IF LARGEST-VALUE-OF-CRITERION IS NOT
LESS THAN RANKING-CRITERION
GO TO READ-SAME-SKILL-GROUP.
MOVE RANKING-CRITERION TO
LARGEST-VALUE-OF-CRITERION.
MOVE OLD-ID-NUMBER TO LARGEST-ID-NUMBER.
MOVE OLD-NAME TO LARGEST-NAME.

READ-SAME-SKILL-GROUP.
READ OLD-PERSONNEL-FILE RECORD AT END
MOVE 1 TO END-OF-PERSONNEL-FILE-SWITCH
GO TO END-OF-SKILL-GROUP.
IF OLD-JOB-SKILL IS NOT EQUAL TO
CURRENT-SKILL-CODE

GO TO END-OF-SKILL-GROUP.
GO TO SELECT-BEST-LOOP.

END-OF-SKILL-GROUP.

MOVE SPACES TO PRINT-LINE.
MOVE JOB-CODE-NUMBER TO PRINT-JOB-CODE.
MOVE JOB-SKILL-REQUIREMENT TO
PRINT-SKILL-REQUIREMENT.
MOVE LARGEST-ID-NUMBER TO PRINT-ID-NUMBER.
MOVE LARGEST-NAME TO PRINT-NAME.
WRITE PRINT-LINE.
GO TO READ-ANOTHER-JOB.

Упражнение

Добавьте или удалите записи файла. Предполагая, что имеется тот же самый файл инвентаризации, описанный в двух предыдущих примерах, напишите программу, которая будет считывать карточные записи и либо добавлять новые записи, либо удалять старые записи из файла инвентаризации, расположенного на ленте. Вводимые перфокарты упорядочены по возрастанию кода товара и, как и раньше, содержат поля для наименования товара, кода поставщика и количества товара в наличии. Кроме того, имеется поле для кода изменения, который может принимать два значения — либо код «добавить запись», либо код «удалить запись». В записях, задающих удаление, поля для наименования поставщика и количества товара будут пустыми. Программа должна прочесть старый файл инвентаризации и сформировать новый файл инвентаризации. Если запись, задающая добавление, обнаруживает запись с таким же идентификационным номером товара или если для записи, задающей удаление, не находится соответствующей записи в файле инвентаризации, то печатается сообщение об ошибке. Для данного примера предположим, что во входном файле на перфокартах нет карт с одинаковыми номерами товаров.

5.5. Поиск по образцу

Программа, рассматриваемая в данном разделе, реализует еще один поисковый процесс в файле и содержит более сложное сравнение, чем единственный вопрос, задаваемый в предыдущем разделе. Рассматриваемая задача заключается в поиске всех записей в файле сотрудников, в которых значения определенных данных равны заданному набору значений. Например, запрос на поиск в файле

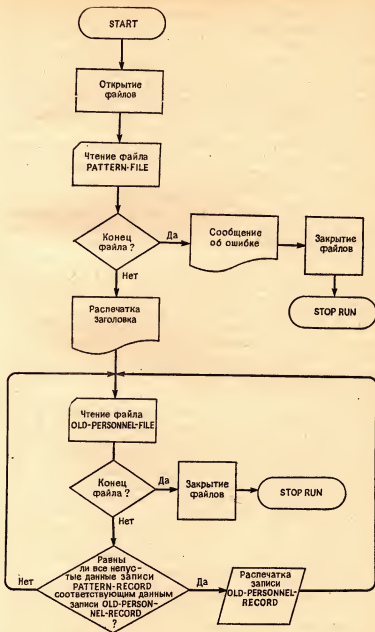


Рис. 5.5. Блок-схема поиска по образцу.

сотрудников может быть задан следующим образом: составить список всех сотрудников в возрасте 45 лет, проживающих в Нью-Йорке с пятью иждивенцами.

Заметим, что в задаче требуется точное совпадение с заданными значениями, а не выполнение неравенств, например «в возрасте от 20 до 45 лет» и «не более чем с пятью иждивенцами». Программист должен четко представлять специфику задачи и скрупулезно учитывать ее при написании программы. Нельзя решать некорректно поставленную задачу, даже если ее правильно запрограммировать. Основные этапы решения рассматриваемой задачи изображены на блок-схеме, представленной на рис. 5.5. На первом шаге обработки считывается карта с запросом на поиск из файла PATTERN-FILE. В данной программе этот файл содержит только одну карту, и только на один запрос дается ответ. Однако требуется внести лишь незначительные изменения по организации цикла с возвратом к параграфу READ-PATTERN-FILE, чтобы программа могла обрабатывать несколько запросов. Формат считываемой записи в точности соответствует формату записей, хранящихся в файле сотрудников. Метод решения заключается в проверке на сравнение полей записи-образца с соответствующими полями последовательно выбираемых записей из файла сотрудников. Если поле данных в записи-запросе заполнено пробелами, то проверка не делается. Если же в нем есть какая-либо информация, то оно проверяется на равенство с соответствующим данным записи-сотрудника. Если будет хотя бы одно несовпадение, то считывается следующая запись-сотрудника. Все непустые данные записи-запроса должны совпасть с соответствующими данными записи-сотрудника перед тем, как последняя будет распечатана. Процедура, реализованная в приведенной программе, не относится к числу наиболее эффективных методов просмотра файла или выполнения операций поиска информации.

В КОБОЛ-программе статьи-описания-записи для записей PATTERN-RECORD и OLD-PERSONNEL-RECORD совпадают. В секции рабочей-памяти имеется только одна статья (в программе HEADING-X), предназначенная для выдачи на печатающее устройство заголовка сообщения. Основную часть раздела PROCEDURE DIVISION составляет последовательность из пятнадцати параграфов, каждый из которых проверяет соответствие между определенными данными. Если в процессе выполнения программы достигается параграф Q-15, то выполняется предложение

WRITE PRINT-LINE FROM OLD-PERSONNEL-RECORD

которое помещает текущую запись из файла сотрудников в выходную область файла PRINT-FILE и распечатывает ее на печатающем устройстве вместе с заголовком сообщения. Программа имеет вид:

IDENTIFICATION DIVISION.

PROGRAM-ID PROG550.

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER. COMPUTER-NAME.

OBJECT-COMPUTER. COMPUTER-NAME.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

 SELECT PATTERN-FILE

 ASSIGN TO READER.

 SELECT OLD-

 PERSONNEL-FILE

 ASSIGN TO TAPE.

 SELECT PRINT-FILE

 ASSIGN TO PRINTER.

DATA DIVISION.

FILE SECTION.

FD PATTERN-FILE

 LABEL RECORDS ARE STANDARD.

01 PATTERN-RECORD.

 05 PATTERN-ID-NUMBER

 PICTURE IS X(10).

 05 PATTERN-NAME

 PICTURE IS X(25).

 05 PATTERN-SEX

 PICTURE IS X.

 05 PATTERN-AGE

 PICTURE IS XX.

 05 PATTERN-HEIGHT

 PICTURE IS X(3).

 05 PATTERN-JOB-SKILL

 PICTURE IS X(8).

 05 PATTERN-ANNUAL-SALARY

 PICTURE IS X(7).

 05 PATTERN-DATE-LAST-

 PROMOTED

 PICTURE IS X(6).

 05 PATTERN-WEIGHT

 PICTURE IS X(3).

 05 PATTERN-HOME-STATE

 PICTURE IS X(2).

 05 PATTERN-DEPENDENTS

 PICTURE IS X(2).

 05 PATTERN-MERIT-RATING

 PICTURE IS X(3).

 05 PATTERN-SENIORITY-RATING

 PICTURE IS X(2).

 05 PATTERN-DATE-HIRED

 PICTURE IS X(6).

FD OLD-PERSONNEL-FILE

 LABEL RECORDS ARE STANDARD.

01 OLD-PERSONNEL-RECORD.

05 OLD-ID-NUMBER	PICTURE IS 9(10).
05 OLD-NAME	PICTURE IS X(25).
05 OLD-SEX	PICTURE IS A.
05 OLD-AGE	PICTURE IS 99.
05 OLD-HEIGHT	PICTURE IS 99V9.
05 OLD-JOB-SKILL	PICTURE IS 9(8).
05 OLD-ANNUAL-SALARY	PICTURE IS 9(5)V9(2).
05 OLD-DATE-LAST- PROMOTED	PICTURE IS 9(6).
05 OLD-WEIGHT	PICTURE IS 9(3).
05 OLD-HOME-STATE	PICTURE IS A(2).
05 OLD-DEPENDENTS	PICTURE IS 9(2).
05 OLD-MERIT-RATING	PICTURE IS 9(3).
05 OLD-SENIORITY-RATING	PICTURE IS 9(2).
05 OLD-DATE-HIRED	PICTURE IS 9(6).

FD PRINT-FILE

LABEL RECORDS ARE STANDARD.

01 PRINT-LINE.

05 MESSAGE-X	PICTURE IS X(50).
05 INPUT-RECORD	PICTURE IS X(80).

WORKING-STORAGE SECTION.

01 HEADING-X PICTURE IS X(50) VALUE IS
 "THE FOLLOWING RECORDS WERE FOUND
 FOR THIS PATTERN".

PROCEDURE DIVISION.

OPEN-FILES-PARAGRAPH.

OPEN INPUT PATTERN-FILE OLD-PERSONNEL-FILE
 OUTPUT PRINT-FILE.

READ-PATTERN-FILE.

READ PATTERN-LINE RECORD AT END GO TO
 NO-INPUT-CARD.
 MOVE HEADING-X TO MESSAGE-X.

MOVE PATTERN-RECORD TO INPUT-RECORD.
WRITE PRINT-LINE.

READ-FILE.

READ OLD-PERSONNEL-FILE RECORD AT END
GO TO CLOSE-FILES-PARAGRAPH.

- Q-1. IF PATTERN-ID-NUMBER IS EQUAL TO SPACES
GO TO Q-2.
IF PATTERN-ID-NUMBER IS NOT EQUAL TO
OLD-ID-NUMBER GO TO READ-FILE.
- Q-2. IF PATTERN-NAME IS EQUAL TO SPACES
GO TO Q-3.
IF PATTERN-NAME IS NOT EQUAL
TO OLD-NAME GO TO READ-FILE.
- Q-3. IF PATTERN-SEX IS EQUAL TO SPACES
GO TO Q-4.
IF PATTERN-SEX IS NOT EQUAL TO OLD-SEX
GO TO READ-FILE.
- Q-4. IF PATTERN-AGE IS EQUAL TO SPACES
GO TO Q-5.
IF PATTERN-AGE IS NOT EQUAL TO OLD-AGE
GO TO READ-FILE.
- Q-5. IF PATTERN-HEIGHT IS EQUAL TO SPACES
GO TO Q-6.
IF PATTERN-HEIGHT IS NOT EQUAL TO
OLD-HEIGHT GO TO READ-FILE.
- Q-6. IF PATTERN-JOB-SKILL IS EQUAL TO SPACES
GO TO Q-7.
IF PATTERN-JOB-SKILL IS NOT EQUAL TO
OLD-JOB-SKILL GO TO READ-FILE.
- Q-7. IF PATTERN-ANNUAL-SALARY IS EQUAL TO
SPACES GO TO Q-8.
IF PATTERN-ANNUAL-SALARY IS NOT EQUAL
TO OLD-ANNUAL-SALARY
GO TO READ-FILE.

- Q-8. IF PATTERN-DATE-LAST-PROMOTED IS EQUAL
TO SPACES GO TO Q-9.
IF PATTERN-DATE-LAST-PROMOTED IS NOT
EQUAL TO OLD-DATE-LAST-PROMOTED
GO TO READ-FILE.
- Q-9. IF PATTERN-WEIGHT IS EQUAL TO SPACES
GO TO Q-10.
IF PATTERN-WEIGHT IS NOT EQUAL
TO OLD-WEIGHT
GO TO READ-FILE.
- Q-10. IF PATTERN-HOME-STATE IS EQUAL TO SPACES
GO TO Q-11.
IF PATTERN-HOME-STATE IS NOT EQUAL TO
OLD-HOME-STATE GO TO READ-FILE.
- Q-11. IF PATTERN-DEPENDENTS IS EQUAL TO
SPACES GO TO Q-12.
IF PATTERN-DEPENDENTS IS NOT EQUAL TO
OLD-DEPENDENTS GO TO READ-FILE.
- Q-12. IF PATTERN-MERIT-RATING IS EQUAL TO
SPACES GO TO Q-13.
IF PATTERN-MERIT-RATING IS NOT EQUAL TO
OLD-MERIT-RATING GO TO READ-FILE.
- Q-13. IF PATTERN-SENIORITY-RATING IS EQUAL
TO SPACES GO TO Q-14.
IF PATTERN-SENIORITY-RATING IS NOT
EQUAL TO OLD-SENIORITY-RATING
GO TO READ-FILE.
- Q-14. IF PATTERN-DATE-HIRED IS EQUAL TO
SPACES GO TO Q-15.
IF PATTERN-DATE-HIRED IS NOT EQUAL TO
OLD-DATE-HIRED GO TO READ-FILE.
- Q-15. WRITE PRINT-LINE FROM OLD-PERSONNEL-
RECORD.
GO TO READ-FILE.

CLOSE-FILES-PARAGRAPH.

CLOSE PATTERN-FILE OLD-PERSONNEL-FILE.
PRINT-FILE.
STOP RUN.

NO-INPUT-CARD.

MOVE "ERROR — MISSING INPUT PATTERN
CARD" TO MESSAGE-X.
MOVE ALL SPACES TO INPUT-RECORD. WRITE
PRINT-LINE.
CLOSE PATTERN-FILE OLD-PERSONNEL-FILE
PRINT-FILE.
STOP RUN.

Упражнение

Просуммировать количество товаров, доставляемых поставщиком. Обновленный файл инвентаризации состоит из записей, которые упорядочены по коду товара (как и в предыдущих примерах) и содержат наименование товара, код поставщика и количество товара в наличии. Требуется написать КОБОЛ-программу для считывания карты, содержащей единственное поле с числовым кодом поставщика, просмотра файла инвентаризации и подсчета общего количества товаров, доставляемых этим поставщиком. Каждый поставщик может поставлять товары разных типов. В задаче требуется подсчитать не количество различных типов доставляемых товаров, а общее количество товаров всех типов, доставляемых указанным поставщиком. Файл упорядочен только по возрастанию кода товара, а не по коду поставщика. Просмотрев файл до конца, напечатайте задаваемый код поставщика и величину общего числа доставляемых им товаров. Составив КОБОЛ-программу для обработки одной вводимой карты, измените программу на случай считывания произвольного числа входных карт. Постарайтесь обойтись без сортировки файла инвентаризации.

5.6. Вычисления и выборочная обработка

В этом разделе предполагается, что имеется файл сотрудников, записи которого располагаются в произвольном порядке. Записи файла считываются последовательно до его исчерпывания. В результате вычисляется величина среднего возраста всех сотрудников и величина среднего годового оклада. Кроме того, печатаются два списка: список записей всех сотрудников, которые не были повышены в должности в течение последних шести месяцев, и список записей сотрудников с годовым окладом менее 5500 долларов.

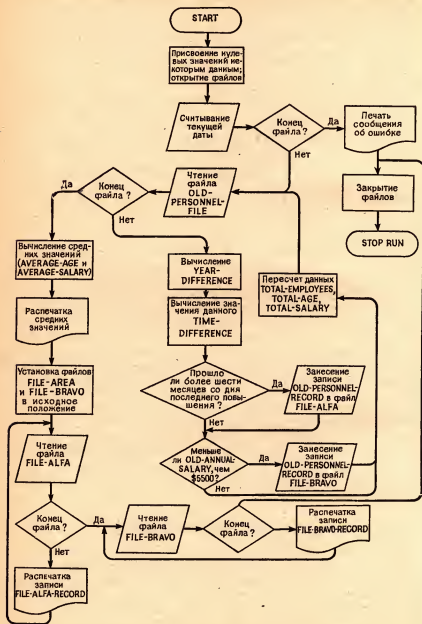


Рис. 5.6. Блок-схема процесса вычислений.

Этот пример программы дает возможность продемонстрировать возможности арифметических операторов и показывает, как можно повторно просматривать файл. В блок-схеме программы, изображенной на рис. 5.6, сначала считывается одна карта, на которой отперфорирована текущая дата (TODAYS-DATE). Это значение является точкой отсчета при выборе сотрудника, который не был повышен в должности за последние шесть месяцев. Затем внутри основного цикла считываются записи файла сотрудников. Для каждой записи вычисляется разница-лет (YEAR-DIFFERENCE), которая затем используется при вычислении выражения:

$$\text{TIME-DIFFERENCE} = 1200 * \text{YEAR-DIFFERENCE} + \text{TODAYS-MONTH-AND-DAY} - \text{MONTH-AND-DAY-OF-LAST-PROMOTION.}$$

(РАЗНИЦА-ВО-ВРЕМЕНИ = 1200 * РАЗНИЦА-ЛЕТ + ТЕКУЩИЙ-МЕСЯЦ-И-ДЕНЬ — МЕСЯЦ-И-ДЕНЬ-ПОСЛЕДНЕГО-ПОВЫШЕНИЯ)

За каждый полный месяц, прошедший со дня повышения, это выражение начисляет число 100 (независимо от числа дней в месяце) и большее или меньшее число, если количество дней не соответствует полному месяцу. В этом способе вычисления месячной протяженности подчеркивается, что важно четко определить, что понимается под шестью месяцами: шесть полных месяцев, 180 дней или двадцать шесть недель? Каждое конкретное определение требует соответствующего способа вычисления.

Если результатом вычисления этого выражения окажется число большее 599 (что означает срок больший, чем шесть месяцев), то запись-сотрудника (OLD-PERSONNEL-RECORD) записывается в файл-А (FILE-ALFA). Затем эта же запись может быть записана в файл-В (FILE-BRAVO), если годовой оклад сотрудника (OLD-ANNUAL-SALARY) меньше 5500 долларов. В конечном итоге в файле-А и файле-В оказываются различные подмножества, состоящие из записей файла-сотрудников. Обработка очередной записи заканчивается рядом операций, с помощью которых увеличивается счетчик числа считываемых записей (TOTAL-EMPLOYEES) и к текущим сумме возраста (TOTAL-AGE) и сумме окладов (TOTAL-SALARY) прибавляются соответственно возраст (OLD-AGE) и оклад (OLD-ANNUAL-SALARY) из очередной записи. Перед началом цикла чтения файла сотрудников в эти переменные были установлены в качестве начальных нулевые значения.

По исчерпыванию файла сотрудников вычисляются значения среднего возраста (AVERAGE-AGE) и оклада (AVERAGE-SALARY) делением итоговых сумм на количество записей в файле. Эти средние значения записываются в файл-печати. После этого производится установка файла А таким образом, чтобы его первая запись была подготовлена к чтению. Это достигается закрытием файла и

его повторным открытием. В результате этих действий катушка магнитной ленты будет перемотана к началу файла. Файл-А затем выводится на печать, и только после этого читается файл-В. Это обеспечивает отдельные распечатки файлов. В конце концов файлы закрываются, и работа программы заканчивается. После завершения программы оператор снимает катушки с результатами. Хранение этих катушек производится в соответствии с инструкцией. Программа на КОБОЛе имеет вид

IDENTIFICATION DIVISION.
PROGRAM-ID. PROG560.

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.
SOURCE-COMPUTER. COMPUTER-NAME.
OBJECT-COMPUTER. COMPUTER-NAME.

INPUT-OUTPUT SECTION.
FILE-CONTROL.

SELECT CARD-FILE	ASSIGN TO READER.
SELECT OLD-PERSONNEL-FILE	ASSIGN TO TAPE.
SELECT FILE-ALFA	ASSIGN TO TAPE.
SELECT FILE-BRAVO	ASSIGN TO TAPE.
SELECT PRINT-FILE	ASSIGN TO PRINTER.

DATA DIVISION.

FILE SECTION.

FD CARD-FILE
LABEL RECORDS ARE STANDARD.

01 CARD-INPUT-RECORD.

05 TODAYS-DATE.	
10 TODAYS-MONTH-AND-DAY	PICTURE IS 9(4).
10 YEAR-OF-TODAYS-DATE	PICTURE IS 9(2).
05 FILLER	PICTURE IS X(74).

FD OLD-PERSONNEL-FILE
LABEL RECORDS ARE STANDARD.

01 OLD-PERSONNEL-RECORD.

05 FILLER	PICTURE IS X(36).
05 OLD-AGE	PICTURE IS 99.
05 FILLER	PICTURE IS X(11).
05 OLD-DATE-LAST-PROMOTED.	
10 MONTH-DAY-OF-LAST-	
PROMOTION	PICTURE IS 9(4).
10 YEAR-OF-LAST-	
PROMOTION	PICTURE IS 9(2).
05 FILLER	PICTURE IS X(18).

FD FILE-ALFA

LABEL RECORDS ARE STANDARD.

01 FILE-ALFA-RECORD PICTURE IS X(80).

FD FILE-BRAVO

LABEL RECORDS ARE STANDARD.

01 FILE-BRAVO-RECORD PICTURE IS X(80).

FD PRINT-FILE

LABEL RECORDS ARE STANDARD.

01 PRINT-LINE PICTURE IS X(132).

WORKING-STORAGE-SECTION.

01 ANSWER-RECORD.

05 FILLER VALUE IS SPACES	PICTURE IS X(35).
05 TOTAL-EMPLOYEES	PICTURE IS 9(9).
05 FILLER VALUE IS SPACES	PICTURE IS X(12).
05 AVERAGE-AGE	PICTURE IS 9(5).9.
05 FILLER VALUE IS SPACES	PICTURE IS X(9).
05 AVERAGE-SALARY	PICTURE IS \$(6).99.

01 TOTAL-AGE PICTURE IS 9(15).

01 TOTAL-SALARY PICTURE IS 9(15).

01 YEAR-DIFFERENCE PICTURE IS 9(3).

01 TIME-DIFFERENCE	PICTURE IS 9(6).
01 MESSAGE-X	PICTURE IS X(19)
VALUE IS "NO TODAY'S DATE CARD".	
01 MESSAGE-Y.	
05 FILLER	PICTURE IS X(20)
VALUE IS "SUMMARY INFORMATION —".	
05 FILLER VALUE IS SPACES	PICTURE IS X(10).
05 FILLER	PICTURE IS X(19)
VALUE IS "NUMBER OF EMPLOYEES".	
05 FILLER VALUE IS SPACES	PICTURE IS X(5).
05 FILLER	PICTURE IS X(11)
VALUE IS "AVERAGE AGE".	
05 FILLER VALUE IS SPACES	PICTURE IS X(5).
05 FILLER	PICTURE IS X(14)
VALUE IS "AVERAGE SALARY".	

PROCEDURE DIVISION.

INITIALIZATION-STEPS.

 MOVE ZERO TO TOTAL-EMPLOYEES TOTAL-AGE
 TOTAL-SALARY.

 OPEN INPUT CARD-FILE OLD-PERSONNEL-FILE
 OUTPUT FILE-ALFA FILE-BRAVO PRINT-FILE.
 READ CARD-FILE RECORD AT END WRITE PRINT-
 LINE FROM MESSAGE-X STOP RUN.

READ-PERSONNEL-FILE.

 READ OLD-PERSONNEL-FILE RECORD AT END
 GO TO PRINT-SUMMARY-REPORT.
 DISPLAY OLD-PERSONNEL-RECORD.

CHECK-FOR-RECENT-PROMOTION.

SUBTRACT YEAR-OF-LAST-PROMOTION FROM YEAR-
OF-TODAYS-DATE GIVING YEAR-DIFFERENCE.

COMPUTE TIME-DIFFERENCE = $1200 * \text{YEAR-}$
DIFFERENCE + TODAYS-MONTH-AND-DAY —
MONTH-DAY-OF-LAST-PROMOTION.

IF TIME-DIFFERENCE IS GREATER THAN 599 WRITE
FILE-ALFA-RECORD
FROM OLD-PERSONNEL-RECORD.

CHECK-FOR-LOW-SALARY.

IF OLD-ANNUAL-SALARY IS LESS THAN 5500 WRITE
FILE-BRAVO-RECORD
FROM OLD-PERSONNEL-RECORD.

ACCUMULATE-RUNNING-SUMS.

ADD 1 TO TOTAL-EMPLOYEES.

ADD OLD-AGE TO TOTAL-AGE.

ADD OLD-ANNUAL-SALARY TO TOTAL-SALARY.

GO TO READ-PERSONNEL-FILE.

PRINT-SUMMARY-REPORT.

DIVIDE TOTAL-EMPLOYEES INTO TOTAL-AGE
GIVING AVERAGE-AGE.

DIVIDE TOTAL-EMPLOYEES INTO TOTAL-SALARY
GIVING AVERAGE-SALARY.

WRITE PRINT-LINE FROM MESSAGE-Y.

WRITE PRINT-LINE FROM ANSWER-RECORD.

MOVE ALL SPACES TO PRINT-LINE WRITE PRINT-
LINE.

PRINT-DETAILED-REPORTS.

CLOSE FILLE-ALFA FILE-BRAVO.

OPEN INPUT FILE-ALFA FILE-BRAVO.

READ-FILE-A.

READ FILE-ALFA RECORD AT END GO TO READ-
FILE-B.

WRITE PRINT-LINE FROM FILE-ALFA-RECORD.
GO TO READ-FILE-A.

READ-FILE-B.

READ FILE-BRAVO RECORD AT END CLOSE FILE-
ALFA FILE-BRAVO
STOP RUN.

WRITE PRINT-LINE FROM FILE-BRAVO-RECORD.
GO TO READ-FILE-B.

В разделе DATA DIVISION описывается статья-описания-записи OLD-PERSONNEL-RECORD, которая именуется только те поля записи, которые используются в данной программе: возраст, годовой-оклад и дата-последнего-повышения. Фраза PICTURE для данного OLD-ANNUAL-SALARY содержит подразумеваемую десятичную точку, но напомним, что она не занимает позиции во внутреннем представлении данного. Поэтому значение данного \$4525.75 было бы размещено в семи литерных позициях следующим образом:

0452575

Запись CARD-INPUT-RECORD содержит шесть цифр, например

102572

что означает: «десятого месяца, двадцать пятого числа, 1972 года». При этом значением данного TODAYS-MONTH-AND-DAY было бы 1025, а значением данного YEAR-OF-TODAY-DATE—72. Размер карточной записи должен равняться восьмидесяти символам, чтобы соответствовать физической записи устройства ввода перфокарт, поэтому последняя статья описания записи имеет вид

FILLER PICTURE X(74)

Остальные три файла имеют простые статьи-описания-записей, так как не требуют никаких подразделений.

Запись ANSWER-RECORD из секции WORKING-STORAGE SECTION предназначена для размещения выходной записи.

В ней предусмотрены интервалы между величинами, которые описаны как FILLER с начальными значениями, состоящими из пробелов. В разделе PROCEDURE DIVISION оператор DIVIDE использует фразу GIVING, которая допускает хранение результата в принимающем поле как цифровое-редактируемое данное. Поэтому напечатанное данное будет иметь валютный знак и явную десятичную точку, вставленную в соответствующую позицию.

Упражнение

Найти максимальное количество товара. В качестве последнего упражнения в серии примеров обработки файла инвентаризации, напишите КОБОЛ-программу, просматривающую файл инвентаризации и печатающую ту запись, которая имеет наибольшее количество товаров в наличии. Если имеется несколько таких записей, то выдайте все их на печать. При этом потребуются сначала просмотреть файл для отыскания величины наибольшего количества товара и затем просмотреть файл еще один раз для отыскания записей, содержащих эту величину.

Глава 6. Описание последовательного файла

6.1. Мнемонические имена

В первых пяти главах описывалось основное подмножество языка КОБОЛ. Это подмножество обеспечивает набор средств, позволяющих начинающему программисту составлять программы для широкого круга задач из области коммерческих применений. Начиная с этой главы, описываются дополнительные средства, позволяющие более гибкое и более сложное использование языка. Помимо введения новых операторов, рассматриваются дополнительные возможности операторов, описанных ранее. Например, оператор **DISPLAY (ВЫДАТЬ)** был уже определен в гл. 5 как

$$\underline{\text{DISPLAY}} \left\{ \begin{array}{l} \text{литерал} \\ \text{имя-данного} \end{array} \right\} \dots$$

Это определение оператора является упрощенным. Полное определение оператора **DISPLAY** имеет вид:

$$\underline{\text{DISPLAY}} \left\{ \begin{array}{l} \text{литерал-1} \\ \text{идентификатор-1} \end{array} \right\} \left[\begin{array}{l} \text{,литерал-2} \\ \text{,идентификатор-2} \end{array} \right] \dots \left[\underline{\text{UPON}} \text{ мнемоническое-имя} \right]$$

В первом определении имеются три отличия от данного в гл. 5: замена имя-данного на идентификатор, разрешение списка (вставка запятых) и добавление фразы **UPON (НА)**. Напомним, что имя-данного является незарезервированным словом КОБОЛа, начинающимся по крайней мере с одной буквенной литеры, которое именуется статью, определенную в разделе **DATA DIVISION**. В некоторых случаях, положение данного, определяемого именем-данного, может быть изменено с помощью уточнения, индексирования с помощью индекса или с помощью имени-индекса этого имени-данного. Эти операции еще будут описаны (см. гл. 7 и 9), поэтому пока нет необходимости касаться этих вопросов. Однако теперь такого рода деталям будет уделяться большее внимание, и иногда в формальных определениях для ссылки на полное имя данного будет использо-

ваться слово *идентификатор*. В настоящий момент и до тех пор, пока слово идентификатор не будет объяснено более полно, это изменение незначительно, не оказывает никакого влияния и просто сводится к текстуальной замене в определениях конструкций фраз слова «имя-данного» на слово «идентификатор», что указывает на несколько другой подход к такому определению в этой части текста.

Наличие запятых в необязательных фразах является еще одним уточнением формальных определений. В некоторые определения будут включены знаки препинания — запятая и точка с запятой. Использование запятых и точек с запятой всегда необязательно, поэтому в тех определениях, в которых они указаны, программист может использовать их или опускать по своему усмотрению. Употребление этих знаков препинания не дает больших преимуществ, однако делает исходную программу более удобочитаемой. Они присущи КОБОЛу, и по этой причине включены в определения языка.

Параграф SPECIAL NAMES

В дополнительной фразе, фразе UPON, содержится мнемоническое имя. Мнемоническим именем может быть любое незарезервированное слово КОБОЛа, определяемое программистом. Смысл этого имени определяется специальной статьей КОБОЛа, записываемой в секции конфигурации раздела оборудования. Эта статья входит в состав впервые упомянутого здесь параграфа SPECIAL-NAMES (СПЕЦИАЛЬНЫЕ-ИМЕНА) и имеет следующий формат:

[имя-реализации IS мнемоническое-имя] . . .

Имя-реализации — это любое допустимое имя КОБОЛа, определяемое автором компилятора (разработчиком языка). В тот момент, когда разработчик включает это имя в состав языка, оно становится зарезервированным словом для этого компилятора. В связи с этим для выполнения исходной КОБОЛ-программы на различных компиляторах обычно требуется внесение изменений в раздел оборудования.

Такое косвенное определение имени-реализации через мнемоническое-имя используется в целях обеспечения программой совместимости на максимально возможном числе различных вычислительных машин. Как упоминалось в предыдущих главах, разработка языка велась таким образом, чтобы раздел оборудования и в меньшей степени раздел данных содержали и изолировали от основной программы те статьи, которые необходимо модифицировать для компиляции исходной КОБОЛ-программы на другой машине. Оператор DISPLAY можно было бы записать в разделе процедур в следующем виде:

DISPLAY ANSWER-X UPON OPERATOR-TYPEWRITER.

Мнемоническое-имя OPERATOR-TYPEWRITER имело бы смысл только в том случае, если в разделе оборудования имелось бы, например, такое описание:

CONFIGURATION SECTION.

. . .

SPECIAL-NAMES.

CONSOLE IS OPERATOR-TYPEWRITER.

В процессе выполнения объектной программы значение данного ANSWER-X было бы выведено на пульт (CONSOLE) вычислительной машины. Для выполнения программы на другой машине, на которой вместо устройства с именем CONSOLE употребляется другое устройство с именем PRINTER, необходимо изменить раздел оборудования исходной программы следующим образом:

CONFIGURATION SECTION.

. . .

SPECIAL-NAMES.

PRINTER IS OPERATOR-TYPEWRITER.

Возможные имена-реализации определяются для каждого конкретного компилятора КОБОЛа его разработчиком, и для каждого компилятора существует свой список этих имен.

Параграф SPECIAL-NAMES используется также с двумя другими операторами процедур, из которых первый определяется впервые, а второй представляет модификацию ранее определенного оператора.

Первый оператор

ACCEPT идентификатор [FROM мнемоническое-имя]

осуществляет передачу информации от физического устройства к данному, задаваемому идентификатором. При этом, как и в случае оператора DISPLAY, для передаваемого данного не требуется статья-описания-данного или какая-либо другая связь с исходной программой, так как оператор ACCEPT (ПРИНЯТЬ) имеет дело с физическим устройством, а не с логическим файлом. Мнемоническое-имя в варианте FROM (ИЗ) должно быть связано с помощью параграфа SPECIAL-NAMES с именем-реализации, которое соответствует некоторому физическому устройству. Если вариант FROM не задан, то по умолчанию используется устройство, которое определяется реализацией как стандартное. Например, таким стандартным устройством обычно является пультовая пишущая машинка. Передаваемая информация размещается в данном слева направо, как при выполнении группового оператора MOVE (ПОМЕСТИТЬ). Если передается меньшее число символов, то данное

дополняется пробелами; в случае числа символов, большего, чем может разместиться в данном, информация усекается.

Операторы ACCEPT и DISPLAY полезно использовать при передаче небольших порций данных, например при отладке программы.

Рассмотрим пример оператора ACCEPT:

ACCEPT SPECIAL-TEST-DATA FROM CONSOLE-UNIT.

Мнемоническое-имя CONSOLE-UNIT должно быть определено предварительно, например так:

SPECIAL-NAMES.

CARD-READER IS CONSOLE-UNIT.

Если бы оператор имел вид

ACCEPT SPECIAL-TEST-DATA

то было бы использовано устройство, определенное реализацией в качестве стандартного. Имя этого устройства можно найти в руководстве для программиста на конкретной машине. Если устройство, определенное рассмотренным выше способом, будет использовано в операторе READ (ЧИТАТЬ), то результаты будут непредсказуемыми и, вероятно, искаженными. Таким образом, для операторов ACCEPT и READ должны использоваться разные входные устройства.

Вторая форма оператора ACCEPT не связана с параграфом SPECIAL-NAMES:

ACCEPT идентификатор FROM $\left\{ \begin{array}{c} \text{DATE} \\ \text{DAY} \\ \text{TIME} \end{array} \right\}$

В этом случае также осуществляется передача значений данных в данное, задаваемое идентификатором. Однако передаваемые данные поступают не с задаваемых внешних устройств, а генерируются в машине по внутренним часам и механизму даты. Для программиста не имеет значения, как работают эти механизмы, фактически на разных машинах они устроены по-разному, но если таковые имеются, то от оператора требуется вводить текущую дату и устанавливать часы каждые сутки. Однако как бы ни работали эти механизмы, при выполнении оператора

ACCEPT CURRENT-TIME-X FROM TIME

фактическое время передается в данное CURRENT-TIME-X. Для зарезервированных слов DATE (ДАТА), DAY (ДЕНЬ) и TIME (ВРЕМЯ) в разделе данных не должно быть каких-либо статей-описания-данных, так как эти слова не порождают данных в памяти

и на них нельзя сослаться. По этой причине следующая конструкция является недопустимой:

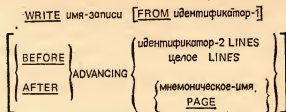
MOVE DATE TO OUTPUT-DATE-ITEM (НЕВЕРНО!)

Значением данного DATE является шестизначное целое число без знака, цифры которого представляют слева направо год текущего столетия, месяц года и число месяца. Таким образом, в годе указываются только две последние цифры. Например, 20 декабря 1974 года было бы представлено числом 741220. Значением данного DAY является пятизначное целое число без знака, две первых цифры которого представляют текущий год, а три последних — порядковый номер дня в году, считая от 1-го января до 31-го декабря, как от 1 до 365 (или 366 в високосном году). Программисты часто называют этот порядковый номер дня года датой Юлианского календаря. В этом случае 20 декабря 1974 года было бы представлено числом 74354. Третье возможное данное — TIME состоит из восьми цифр, которые представляют часы, минуты, секунды и сотые доли секунды. При отсчете времени учитывается двадцатичетырехчасовая продолжительность суток. Отсчет начинается от полуночи, когда значение данного TIME равно 00000000, и до следующей полуночи, когда значение данного TIME становится равным 23595999. После двенадцати часов дня времена 1 : 00, 2 : 00 и 3 : 00 будут порождать значения данного TIME, соответственно равные 13, 14 и 15 часам. Например, ровно двадцать минут девятого после полудня будет представлено значением 20200000. В некоторых вычислительных машинах время отсчитывается на шестидесятигерцевой основе и каждая 1/60 секунды преобразуется в ближайшее десятичное приближение.

Фактические значения данных DATE, DAY и TIME постоянно изменяются: данное TIME — каждую сотую долю секунды, а данные DATE и DAY — каждый раз в полночь. Программист может использовать эту текущую информацию для датирования выданных, для занесения даты и времени генерации в выходные файлы, для подсчета времени работы и для других целей. Например, последний пример задачи из гл. 5 можно было бы значительно упростить, используя датирование в виде порядкового номера дня в году.

Расширенный оператор WRITE

Ранее введенный оператор WRITE (ПИСАТЬ) также имеет дополнительные возможности, задаваемые фразой ADVANCING (ПРОДВИЖЕНИЯ). Теперь расширенное определение оператора таково:



Рассмотрим примеры оператора:

WRITE OUTPUT-RECORD.

WRITE OUTPUT-RECORD FROM STORED VALUE.

WRITE OUTPUT-RECORD AFTER ADVANCING A-ITEM LINES.

WRITE OUTPUT-RECORD FROM STORED-VALUE BEFORE
ADVANCING PAGE.

WRITE OUTPUT-RECORD AFTER ADVANCING
SPECIAL-NAME.

Фраза ADVANCING управляет вертикальным позиционированием строки (движением бумаги), когда для выходного файла назначено печатающее устройство. Вертикальное позиционирование бумаги в печатающем устройстве аналогично возврату каретки на печатающей машинке; бумага продвигается только вперед, так как на печатающем устройстве невозможно вернуть или перемотать бумагу назад. В каждой записи файла, ориентированного на печатающее устройство, независимо от того, подключено оно к машине или будет использовано автономно, должна содержаться информация о продвижении бумаги. Обычно в выходной записи для этих целей отводится первая символьная позиция, в которой хранится код, управляющий перемещением бумаги в момент печати. Программисту не обязательно знать значение кода, управляющего движением бумаги, но в статье-описания-данных раздела данных для этого кода должна быть зарезервирована символьная позиция, например так, как это сделано в следующих статьях описания:

01 OUTPUT-RECORD.

05 FILLER

PICTURE IS X.

05 REST-OF-RECORD

PICTURE IS X(120).

Возможно, что в некоторых компиляторах это поле не будет использоваться, однако в целях совместимости его следует включать в описание записи. О совместимости упоминается уже не в первый раз в связи с тем, что нельзя допускать, чтобы исходные программы, составляющие основной капитал компании, были пленниками какой-то одной реализации.

Во фразе ADVANCING слово LINES (СТРОК) не является обязательным, но если оно используется, то должно быть только в том виде, в котором оно представлено в формате. Таким образом, для продвижения даже одной строки необходимо, чтобы оператор имел вид:

WRITE OUTPUT-RECORD AFTER ADVANCING 1 LINES.

Если для файла применяется фраза ADVANCING, то во всех операторах WRITE для заданного файла эта фраза должна присутствовать. Варианты идентификатор-2 или целое определяют числовое элементарное данное, значением которого является целое неотрицательное число, и служат для непосредственного задания значения кода движения бумаги. Диапазон значений этого числа от 0 до 99 включительно.

Приведем несколько примеров употребления оператора WRITE:

WRITE OUTPUT-RECORD AFTER ADVANCING 15 LINES.

WRITE PRINT-FILE-RECORD FROM WORK-RECORD
BEFORE ADVANCING NUMBER-X LINES.

Если фраза ADVANCING отсутствует, то после печати пропускается одна строка.

Если в операторе используется мнемоническое-имя, то с помощью параграфа SPECIAL-NAMES оно должно быть связано с некоторым именем-реализации, которое определяет функцию пропуска-строк. Как и раньше, эти имена определяются для каждой машины по-своему. Функции пропуска-строк могут либо запрещать продвижение бумаги, либо подводить бумагу к некоторой определенной строке страницы и выполняются механизмом управления движения бумаги печатающего устройства. Например, бумага может быть продвинута к началу следующей страницы при условии, что в программе включены следующие описания:

(1) в разделе оборудования:

SPECIAL-NAMES.

CO 1 IS TO-NEW-PAGE.

(2) в разделе процедур:

WRITE PRINT-LINE-RECORD AFTER ADVANCING
TO-NEW-PAGE.

На печатающем устройстве вычислительной машины не допускается прогон бумаги назад. Поэтому при печати невозможно вернуться обратно. Следовательно, значением вариантов идентификатор-2 или целого никогда не может быть отрицательное число. Однако и в том, и в другом случае может быть нулевое значение, что

вызывает запрещение (блокировку) перевода строк. Таким образом, наложение печатаемых строк может быть осуществлено несколькими способами. Это может быть сделано с помощью оператора WRITE,

```
IDENTIFICATION DIVISION.  
PROGRAM-ID.    PROG810.
```

```
ENVIRONMENT DIVISION.
```

```
CONFIGURATION SECTION.  
SOURCE-COMPUTER.    COMPUTER-NAME.  
OBJECT-COMPUTER.    COMPUTER-NAME.
```

```
SPECIAL-NAMES.  
SYSIN IS OPERATOR-INPUT  
CO1 IS TO-NEW-PAGE.
```

```
INPUT-OUTPUT SECTION.  
FILE-CONTROL.  
    SELECT PRINT-FILE    ASSIGN TO UT-S-SYSOUT.
```

```
DATA DIVISION.
```

```
FILE SECTION.  
FD PRINT-FILE    LABEL RECORD IS STANDARD.  
01 PRINT-LINE    PICTURE IS X(120).
```

```
WORKING-STORAGE SECTION.
```

```
01 DATA-RECORD.  
    05 FILLER    VALUE IS SPACE    PICTURE IS X.  
    05 DATA-VALUE    PICTURE IS X(30).
```

```
PROCEDURE DIVISION.
```

```
PARA-1.  OPEN OUTPUT PRINT-FILE.  
    ACCEPT DATA-VALUE FROM OPERATOR-INPUT.  
    WRITE PRINT-LINE FROM DATA-RECORD AFTER ADVANCING TO-NEW-PAGE.  
    STOP RUN.
```

Рис. 6.1. Пример мнемонических-имен.

не использующего вариант ADVANCING, который печатает запись, после чего продвигает бумагу. Аналогичная работа выполняется оператором WRITE BEFORE ADVANCING (ПИСАТЬ ДО ПРОДВИЖЕНИЯ). Кроме того, имеется возможность вообще запретить продвижение бумаги с помощью функции, определяемой реализацией в параграфе SPECIAL-NAMES, либо оператором, подобным приведенному ниже:

WRITE OUTPUT-RECORD AFTER ADVANCING 0 LINES.

На рис. 6.1. приводится пример использования мнемонических-имен.

6.2. Описание файла

До сих пор статья-описания-файла, записываемая в секции файлов раздела данных, использовалась исключительно для определения, являются ли метки записей стандартными или нет. Фраза LABEL (МЕТКИ) является единственной обязательной фразой в статье FD (ОФ), поэтому включение ее было необходимо, чтобы можно было компилировать КОБОЛ-программу. Как уже упоминалось, обычно эта фраза записывается в виде

LABEL RECORDS ARE STANDARD

и означает, что помимо записей данных в файле имеются две стандартные записи меток, одна из которых предшествует, а вторая замыкает записи файла. Иногда фраза о метках записывается в виде

LABEL RECORDS ARE OMITTED

что необходимо в случаях, когда либо устройство, назначенное файлу (например, устройство ввода перфокарт), определено в реализации как устройство, которое не имеет меток, либо для файла назначена магнитная лента без меток (обычно такая ситуация связана с использованием магнитных лент с другой машины). Записи меток, как правило, не доступны программисту, т. е. КОБОЛ не предоставляет возможность произвольно управлять считыванием или записью этих меток.

Несмотря на это, записи меток играют очень важную роль в обработке файлов. При создании выходного файла с помощью оператора открытия файла OPEN OUTPUT (ОТКРЫТЬ ВЫХОДНОЙ), который предшествует операторам WRITE, создаются записи меток, содержащие информацию о моменте создания и размере файла. Когда впоследствии этот же файл открывается как входной с помощью оператора OPEN INPUT (ОТКРЫТЬ ВХОДНОЙ) и читается с помощью операторов READ, информация, содержащаяся в записях-метках, анализируется с целью контроля правильности выбора файла. Эта работа выполняется автоматически операционной системой, о которой упоминалось в гл. 2. При этом используются определенные сведения, задаваемые на командном языке управления, предназначенном для связи с операционной системой. Ниже приводится состав информации, которая хранится в записях стандартных меток:

1. Идентификационный литерал — для файла он является именем файла (ID). Это имя используется во фразе VALUE OF (ЗНАЧЕНИЕ) для идентификации конкретного файла.

2. Дата создания — день, месяц и год создания файла. В процессе обработки файлов сразу несколько из них могут иметь одинаковое значение ID, но различаться по дням создания.

3. Номер цикла обработки — порядковый номер, начиная с 001, идентифицирующий момент создания файла в течение дня; если цикл обработки файла меньше двадцати четырех часов, с помощью номера цикла достигается единственность идентификации файлов.

4. Дата годности — день, месяц и год, определяющие время жизни файла. Эта дата устанавливается при создании файла с помощью командного языка, и файл никогда не должен использоваться позже этой даты. За этим следит операционная система. Если программа попытается прочитать файл с просроченной датой годности, то операционная система выдаст оператору сообщение об ошибке.

5. Номер катушки магнитной ленты или диска — дополнительное средство для опознания файла в машинах. Обычно катушкам лент и дисковым пакетам присваиваются номера. Дополнительная проверка, как правило, используется в процессе обработки данных для большей надежности.

6. Пароль — секретное слово, которое обеспечивает доступ к файлу только для пользователей, имеющих соответствующие полномочия. Пароль сравнивается со словом, которое вводится оператором с помощью командного языка управления в задании на исполнение программы.

7. Число устройств — оно отлично от 1, если для размещения всего файла требуется более одной катушки с лентой или более одного пакета магнитных дисков; в случае числа устройств больше 1 конец ленты или диска не является концом файла.

8. Номер устройства — порядковый счетчик, обеспечивающий установку нескольких устройств, если это необходимо для файла. В случае неправильной установки катушки или дискового пакета оператору выдается сообщение об ошибке.

9. Счетчик записей — число, помещаемое в запись метки конца файла, которое проверяется по исчерпанию файла для определения: все ли записи были обработаны. Если файл закрывается раньше, чем будет исчерпан, этот контроль не производится. И снова счетчик записей используется для одной из многих проверок, которым подвергается файл в процессе выполнения программы.

В целях более тщательного управления обработкой информации в конкретной задаче во фразе LABELS (МЕТКИ) следует всегда использовать вариант STANDARD (СТАНДАРТНЫЙ). Для того чтобы поддержать эту рекомендацию, в большинстве установок допускается, а иногда и требуется, использование варианта STANDARD даже для устройства ввода с перфокарт.

Статья FD (ОФ) секции файлов, кроме фразы LABELS, может содержать и другие фразы. Они изображены на рис. 6.2. Эти фразы

не обязательны, но если присутствуют, то должны быть правильно специфицированы.

Первой из них является фраза VALUE OF. Она уже упоминалась в гл. 2 как обязательная фраза для некоторых установок, хотя по формату языка она не является обязательной. Эта фраза не тож-

1. Индикатор-уровня-описания-файла :

FD ИМЯ-ФАЙЛА

2. LABEL фраза:

<u>LABEL</u>	<u>RECORD IS</u> <u>RECORDS ARE</u>	<u>STANDARD</u> <u>OMITTED</u>
--------------	--	-----------------------------------

3. VALUE OF φ PRAZA:

VALUE OF имя-данного-3 IS {литерал-1} ...

4. DATA фраза:

DATA { RECORD IS
 RECORDS ARE } имя-записи-1 [, имя-записи-2] ...

5. RECORD фраза:

RECORD CONTAINS [целое-1 TO] целое-2 CHARACTERS

6. BLOCK фраза:

BLOCK CONTAINS [целое-3 TO] целое-4 { RECORDS
 CHARACTERS }

Рис. 6.2. Статья-описания-файла.

дественна фразе VALUE IS, которая используется в статье-описания-данных секции WORKING-STORAGE SECTION для установления начальных значений данных. Фраза VALUE OF не имеет никакого отношения к фразе VALUE IS. Фраза VALUE OF используется в статье-описания-файла для присвоения литерального значения данному, расположенному в стандартной записи метки этого файла. Приведем пример использования фразы VALUE OF:

FD IN-FILE

LABEL RECORDS ARE STANDARD

VALUE OF IDENTIFICATION IS "INVOICE-FILE".

Имя-данного из основного формата определяется в реализации и обычно это либо слово IDENTIFICATION, либо слово ID. Значением литерала должно быть допустимое слово КОБОЛа, длина которого может ограничиваться на конкретных установках. Когда

файл, именуемый словом, стоящим за индикатором уровня FD (в предыдущем примере файл именуется словом IN-FILE), открывается как выходной (OUTPUT), значение литерала из фразы VALUE OF с помощью операционной системы переписывается в запись метки файла в качестве имени файла. Когда затем этот же файл будет использоваться в другой программе как входной (INPUT), то данное в записи метки сравнивается с литералом во фразе VALUE OF. Если их значения не совпадают, происходит обращение к программе ошибок, которая обычно прекращает выполнение программ, тем самым предотвращая обработку неверно установленного файла. В некоторых реализациях отслеживание имени файла производится по другому правилу. Литерал для сравнения с именем читаемого файла вводится в машину посредством командного языка управления. В этом случае фраза VALUE OF игнорируется, и ее наличие не препятствует выполнению.

Третьей фразой, изображенной на рис. 6.2, является фраза DATA RECORDS (ЗАПИСИ ДАННЫХ), которая используется только в целях документируемости для указания имен записей данных. Она не является обязательной, но ее использование весьма желательно в целях улучшения документации исходной КОБОЛ-программы. Хорошая документируемость является сильной стороной КОБОЛа, и если для студента, выполняющего отдельные учебные упражнения, такого рода преимущества могут быть не очевидны, то они в полной мере признаются программистами, постоянно связанными с эксплуатацией и модификацией рабочих программ. Введение на этой стадии фразы DATA RECORDS предоставляет также прекрасную возможность определить здесь понятие записей данных различных типов для одного файла. Для файла нет необходимости вводить ограничение, чтобы все записи, которые он содержит, имели одинаковый формат или даже длину. Для каждого типа записи должна быть своя статья-описания-записи и уникальное имя-записи. Эти различные имена-записей перечисляются в операндах к фразе DATA RECORDS. Рассмотрим пример употребления этой фразы:

FD INPUT-FILE

LABEL RECORDS ARE STANDARD
VALUE OF ID IS "INREC"

DATA RECORDS ARE STUDENT-NAME-RECORD
COURSE-RESULT-RECORD.

01 STUDENT-NAME-RECORD.

05 STUDENT-NUMBER

PICTURE IS X(8).

05 STUDENT-NAME

PICTURE IS X(30).

01 COURSE-RESULTS-RECORD.

- | | |
|------------------|-------------------|
| 05 COURSE-NUMBER | PICTURE IS 9(5). |
| 05 COURSE-NAME | PICTURE IS A(15). |
| 05 COURSE-GRADE | PICTURE IS 9(3). |

Порядок, в котором имена-записей перечислены во фразе DATA RECORDS или в котором записываются статьи-описания-записей, не существует. Также не существует порядок, в котором записи содержатся в файле. Все записи, независимо от типа, поочередно считываются в одну и ту же область памяти, и в каждый отдельный момент выполнения программе может быть доступна только одна запись. Записи, содержащиеся в файле, не имеют имен, и, когда выполняется оператор READ, непосредственно следующая запись из файла помещается в область записи файловой области. Оператор READ ссылается только на имя-файла:

READ INPUT-FILE RECORD AT END GO TO END-PARA.

Программист сам с помощью программных средств должен определить, к какому типу относится очередная запись в области записи файловой области. Один из способов заключается в том, чтобы записи в файле записывались в некотором заранее определенном порядке. В предыдущем примере файл INPUT-FILE можно было бы организовать таким образом, чтобы за каждой записью STUDENT-NAME-RECORD следовали ровно четыре записи COURSE-RESULT-RECORD. Однако такой подход обычно не используется, так как он ограничивает применение файла. Гораздо более удобный и гибкий способ состоит в предусмотрении в каждой записи специального данного, содержащего идентификационный код типа записи. Относительное положение этого данного в записи каждого типа должно быть одинаковым, чтобы программа могла всегда его выбрать и определить тип записи. Записи файла, рассматриваемого нами в качестве примера, изменятся следующим образом:

01 STUDENT-NAME-RECORD.

- | | |
|-------------------|-------------------|
| 05 RECORD-ID-CODE | PICTURE IS X. |
| 05 STUDENT-NUMBER | PICTURE IS X(8). |
| 05 STUDENT-NAME | PICTURE IS X(30). |

01 COURSE-RESULT-RECORD.

- | | |
|------------------------|-------------------|
| 05 IDENTIFICATION-CODE | PICTURE IS X. |
| 05 COURSE-NUMBER | PICTURE IS 9(5). |
| 05 COURSE-NAME | PICTURE IS A(15). |
| 05 COURSE-GRADE | PICTURE IS 9(3). |

Заметим, что имена-данных для кода типа записи в двух статьях-описания-записей различны. Каждое имя-данного должно быть уникальным. На рис. 6.3 показаны относительные положения данных в обеих статьях. И то и другое описание накладывается на одну и ту же область памяти, длина которой устанавливается компилятором равной наибольшей по размеру записи данного файла.

В разделе процедур с помощью оператора IF можно всегда определить, к какому типу записей относится очередная запись: RECORD-ID-CODE или IDENTIFICATION-CODE, например, способом, указанным ниже:

READ-PARAGRAPH.

```
READ INPUT-FILE RECORD AT END GO TO END-PARA.  
IF RECORD-ID-CODE IS EQUAL TO "S"  
    GO TO PROCESS-STUDENT-RECORD.  
IF RECORD-ID-CODE IS EQUAL TO "C"  
    GO TO PROCESS-COURSE-RECORD.  
GO TO ERROR-IN-ID-CODE.
```

Четвертой фразой в статье-описания-файла является фраза RECORD CONTAINS (В ЗАПИСИ), которая указывает размер записей, содержащихся в файле. Размеры записей меток не учитываются в этой фразе, так как эти записи не доступны программисту. Если все записи данных имеют одинаковый размер, то фраза будет похожа на приведенную ниже:

RECORD CONTAINS 100 CHARACTERS

Для записей разных размеров может быть задана такая фраза:

RECORD CONTAINS 26 TO 88 CHARACTERS

в которой указываются минимальный и максимальный размеры записей. Фраза RECORD CONTAINS не является обязательной, но если она присутствует, она определяет размер обрабатываемых записей и будет учитываться в ходе процесса обработки. Кроме того, разумно использовать задание длины для проверки правильности подсчета символов, производимого программистом. Любое несоответствие между целым значением во фразе RECORD CONTAINS и числом символов в записи, вычисленным по статье-описания-записи, будет выявлено либо на этапе компиляции, либо при отладке программы. После рассмотрения всех возможностей файл INPUT-FILE может быть описан следующим образом:

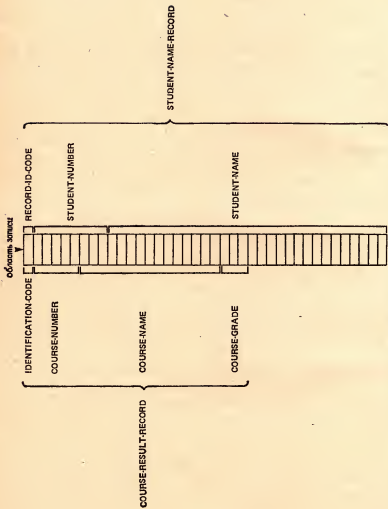


Рис. 6.3. Размещение области записи в файловой области.

FD INPUT-FILE

LABEL RECORDS ARE STANDARD

VALUE OF ID IS "INREC"

DATA RECORDS ARE STUDENT-NAME-RECORD

COURSE-RESULT-RECORD

RECORD CONTAINS 24 TO 39 CHARACTERS.

01 STUDENT-NAME-RECORD.

05 RECORD-ID-CODE

PICTURE IS X.

05 STUDENT-NUMBER

PICTURE IS X(8).

05 STUDENT-NAME

PICTURE IS X(30).

01 COURSE-RESULT-RECORD.

05 IDENTIFICATION-CODE

PICTURE IS X.

05 COURSE-NUMBER

PICTURE IS 9(5).

05 COURSE-NAME

PICTURE IS A(15).

05 COURSE-GRADE

PICTURE IS 9(3).

Последняя фраза в статье FD управляет размером физической записи в файле. Фраза BLOCK CONTAINS (В БЛОКЕ) не является обязательной, но ее следует использовать для указания размера

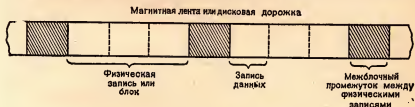


Рис. 6.4. Объединение логических записей в блоки.

физических записей на носителе, используемом для хранения логических записей файла. В этом контексте физическая запись на носителе называется *блоком*. В блоке может содержаться как одна, так и несколько записей данных, значения данных целое-3 и целое-4 на рис. 6.2 управляют этим. На рис. 6.4 схематически изображен характер размещения записей данных вдоль участка магнитной ленты или дисковой дорожки. Несколько записей данных, разделяемых на рисунке вертикальными пунктирными линиями, объединены в одну непрерывную физическую запись (блок). Внутри блока между записями данных нет физических интервалов. Однако блоки на носителе отделяются один от другого межблочными промежутками. Поэтому блок можно определить как область носителя между

двумя межблочными промежутками. Эти промежутки необходимы, так как они приводят к временным задержкам, в течение которых устройства управления вводом-выводом производят действия, требуемые между двумя операциями обмена.

Когда выполняется первый оператор READ, блок целиком передается от носителя файла (лента или диск) в область памяти, назы-

```

FD EXAMPLE-FILE
  LABEL RECDRDS ARE STANDARD
  VALUE OF ID IS "EXAMPLE"
  DATA RECORDS ARE RECORD-TYPE-ONE
    RECORD-TYPE-TWO
  RECDRD CONTAINS 20 CHARACTERS
  BLOCK CONTAINS 200 CHARACTERS.

01 RECORD-TYPE-ONE.
  05 FIELD-A      PICTURE IS X(10).
  05 FIELD-B      PICTURE IS X(10).

01 RECORD-TYPE-TWO.
  05 ITEM-X       PICTURE IS X(7).
  05 ITEM-Y       PICTURE IS X(8).
  05 ITEM-Z       PICTURE IS X(5).
```

Рис. 6.5. Пример организации файла.

ваемую *буфером*. Затем первая запись данных из блока пересылается в область записи из файловой области, которая находится во внутренней памяти. При выполнении второго оператора READ уже нет необходимости снова обращаться к внешнему устройству, назначенному файлу. Очередная запись уже находится в буфере и может быть с большей скоростью передана из него в область записи. Быстрая передача записей из буфера в область записи продолжается до тех пор, пока не будут прочитаны все записи из первого блока. Затем следующий оператор READ вызывает передачу из внешнего устройства в буфер второго блока, и внутренние пересылки в область записи продолжают. Чем больше размер блока, тем выше фактическая скорость ввода-вывода записей. Казалось бы, блоки следует делать очень большими. Но чем больше размер блока, тем больший объем внутренней памяти используется для хранения блока в буфере. Память не может быть безграничной и обычно занята программой, областью рабочей памяти и т. д. Для определения оптимального размера блока не существует простых методов, однако оценить размер блока можно с помощью тестов или на основании знаний характеристик используемого оборудования вычислительной системы.

Фраза BLOCK может быть опущена, если в блоке размещается ровно одна запись данных или если физическое устройство (например, устройство ввода перфокарт) имеет только один допустимый размер физической записи, или, например, если в реализации был опреде-

лен стандартный физический размер записи (например, количество символьных позиций устройства).

Если все записи данных имеют одинаковый размер, то размер блока просто определяется либо фразой:

BLOCK CONTAINS целое CHARACTERS

либо фразой:

BLOCK CONTAINS целое RECORDS

в варианте CHARACTERS (ЛИТЕРЫ) необязательное слово CHARACTERS может быть опущено. Из двух приведенных форм фразы BLOCK первая предпочтительнее. Фактически, существуют определенные ситуации, в особенности характерные для дисковых устройств, когда вариант RECORDS не может быть употреблен. В таких случаях программисту следует обратиться к справочному пособию по использованию КОБОЛа на его машине.

Если записи данных имеют неодинаковый размер, то можно использовать фразу BLOCK в следующей форме:

BLOCK CONTAINS целое-1 TO целое-2 { CHARACTERS
RECORDS }

Целое-1 задает минимальный размер блока, а целое-2 — максимальный. Для каждой реализации существует свой метод определения размера блока для файлов, содержащих записи данных разной длины, поэтому опять-таки предпочтительней использовать вариант CHARACTERS.

6.3. Размещение файла

В предыдущих главах секция ввода-вывода раздела оборудования рассматривалась как содержащая только один параграф — управление-файлом. Этот параграф был определен как последовательность статей вида:

SELECT имя-файла ASSIGN TO имя-устройства

Эти статьи устанавливают связь между программным именем файла и физическим устройством вычислительной машины, на котором размещается логический файл. В статье SELECT могут употребляться и другие варианты, которые приведены ниже:

FILE-CONTROL.SELECT имя-файла ASSIGN TO имя-устройства

[<u>RESERVE</u> целое	[AREA]
			AREAS	
]	
[<u>ORGANIZATION IS SEQUENTIAL</u>			
[<u>ACCESS MODE IS SEQUENTIAL</u>			
[<u>FILE STATUS IS</u> имя-данного			

Теперь статья SELECT могла бы иметь такой вид:

FILE-CONTROL.

SELECT IN-FILE ASSIGN TO MAGNETIC-TAPE
RESERVE 3 AREAS
ORGANIZATION IS SEQUENTIAL
ACCESS MODE IS SEQUENTIAL
FILE STATUS IS IN-FILE-STATUS-CODE.

Все части статьи SELECT должны начинаться на бланке в поле В. Фраза ASSIGN является обязательной. Все остальные фразы не обязательны.

Фраза RESERVE (РЕЗЕРВИРОВАТЬ) характеризует буферные области, используемые во внутренней памяти для ускорения процесса ввода-вывода. Физический блок считывается в область буфера. Затем каждый раз, когда выполняется оператор READ, очередная логическая запись делается доступной программе. Если бы было два таких буфера, то в один можно было бы считывать физический блок, в то время как из другого велась бы пересылка записей данных в область записи. Чередувание работы двух буферов позволило бы значительно убыстрить операции ввода. Увеличение количества буферов для каждого файла позволило бы повысить скорость ввода не только потому, что передача внешних данных занимает миллисекунды, в то время как внутренние передачи исчисляются микросекундами, но еще и потому, что передача от файла к буферу может выполняться параллельно с выполнением внутренних операций. Таким образом, в тот момент, когда второй буфер будет заполняться новым блоком, в первом буфере записи данных становятся доступными программе. Чем больше буферов, тем быстрее выполняются операции, однако для буферов требуется значительная внутренняя память. Таким образом, в этом случае мы сталкиваемся с такой же проблемой, как и при выборе оптимального размера блока. Программист может варьировать количество буферов с помощью фразы

RESERVE целое { AREA }
 { AREAS }

где целое должно быть ненулевым положительным числом. Фраза не обязательная и, если отсутствует, то операционная система сама назначает файлу определенное число буферов. Прием, при котором значение параметра, определенное в реализации, подставляется там, где программист опустил значение параметра, называется *заданием (вариантом) по умолчанию*. Этот прием используется довольно часто и позволяет программисту на КОБОЛе переложить на машину выработку решений по многим вопросам. Это, конечно, удобно, но подробное описание размещения файла в конкретной программе чаще всего приводит к получению более эффективных рабочих программ, нежели при использовании варианта по умолчанию. Если фраза `RESERVE AREAS` отсутствует, то чаще всего в машинах по умолчанию каждому файлу назначаются два буфера. Даже если программист желает воспользоваться вариантом по умолчанию, в целях улучшения документации программы желательно все же явно указывать количество буферов (или областей), назначаемых файлу, как в приводимом примере:

FILE-CONTROL.

```
SELECT INPUT-FILE
      ASSIGN TO MAGNETIC-TAPE-UNIT
      RESERVE 2 AREAS.
SELECT OUTPUT-FILE
      ASSIGN TO TAPE-UNIT-1
      RESERVE 6 AREAS.
SELECT PERSONNEL-FILE
      ASSIGN TO TAPE
      RESERVE 1 AREA.
```

Следующими двумя фразами, которые не обязательны в параграфе `FILE-CONTROL`, являются фразы:

```
ORGANIZATION IS SEQUENTIAL
(ОРГАНИЗАЦИЯ ПОСЛЕДОВАТЕЛЬНАЯ)
ACCESS MODE IS SEQUENTIAL
(ДОСТУП ПОСЛЕДОВАТЕЛЬНЫЙ)
```

На этой стадии изучения КОБОЛа эти фразы не имеют вариантов и, если бы сами фразы отсутствовали, то по умолчанию организация файла выбиралась бы последовательной и доступ к записям на файле тоже был бы последовательным. Однако на самом деле существуют и другие типы организации файла и доступа к нему, и наступило время ознакомить читателя с подробностями описания обслуживания в КОБОЛе.

Записи файлов, используемых в КОБОЛе, могут иметь три типа организации: последовательную, относительную и индексную. Эти типы организации файла имеют дело с неизменной структурой логического файла, определенной в момент создания файла. Три типа организации файла есть не что иное, как три различных способа идентификации записей файла. *Индексный* и *относительный* типы организации будут описаны в гл. 10, но кратко их можно охарактеризовать следующим образом. В индексном файле любая отдельная запись идентифицируется с помощью одного или нескольких данных, которые содержатся в самой записи и образуют ключ записи. Например, если бы в каждой записи файла имелось данное с именем SOCIAL-SECURITY-NUMBER, то можно было бы идентифицировать каждую запись путем указания уникального конкретного значения этого данного. При относительной организации каждая запись идентифицируется целым числом, которое определяет порядковый номер записи в файле, например первая, сороковая, пятьдесят-первая и т. д. Порядковый номер записи не содержится в ней самой. Эти два типа организации файла не приемлемы для магнитной ленты. Только дисковые пакеты обладают необходимыми физическими особенностями для относительной или индексной организации. Все устройства допускают последовательную организацию файла, при которой отдельные записи идентифицируются только отношением типа предшественник-преемник. Единственный способ идентификации записи в такой последовательности состоит в выборе либо *первой*, либо *следующей* записи. Файлом с последовательной организацией являются либо перфокарты в колоде, либо записи на магнитной ленте, следующие друг за другом. Благодаря такой физической природе перфокартных массивов, лент и строк печатающего устройства, все размещенные на них файлы могут иметь только последовательную организацию.

При последовательной организации (ORGANIZATION IS SEQUENTIAL) возможен только один способ доступа к записям файла — последовательный доступ. Таким образом, в простейшем случае наличие двух фраз вместе избыточно, так как из фразы ORGANIZATION IS SEQUENTIAL следует, что доступ тоже последовательный (ACCESS MODE IS SEQUENTIAL). Однако в общем случае существуют и другие виды доступа, которые могут использоваться при способах организации, отличных от последовательного.

Последней фразой в статье SELECT является фраза:

FILE STATUS IS имя-данного
(СОСТОЯНИЕ ФАЙЛА)

которая сообщает системе ввода-вывода вычислительной машины имя-данного, выбранного для помещения признака результата опе-

рации ввода или вывода для файла, имя которого указано в статье SELECT. Имя-данное должно быть описано в секции рабочей памяти как данное, состоящее из двух буквенно-цифровых символов. Таким образом, полная статья SELECT в параграфе FILE-CONTROL из секции INPUT-OUTPUT раздела ENVIRONMENT DIVISION может иметь такой вид:

FILE-CONTROL.

SELECT INPUT-FILE

ASSIGN TO MAGNETIC-TAPE

ORGANIZATION IS SEQUENTIAL

ACCESS MODE IS SEQUENTIAL

RESERVE 3 AREAS

FILE STATUS IS IN-FILE-STATUS-CODE.

Заметим, что порядок следования фраз не имеет значения. Напомним, что для данного примера необходима статья-описания-данного IN-FILE-STATUS-CODE, выделенного под признак результата операции ввода-вывода. Когда выполняется один из операторов CLOSE, OPEN, READ или WRITE, относящийся в нашем примере к файлу с именем INPUT-FILE, то в данное IN-FILE-STATUS-CODE помещается двухсимвольный признак результата этой операции. Значения этих двух символов могут быть проверены в программе с помощью оператора IF для определения результата выполнения операции ввода или вывода. Первый символ называется ключом состояния 1, а второй — ключом состояния 2. Соответствующая статья в секции рабочей памяти должна иметь для нашего примера такое описание:

WORKING-STORAGE SECTION.

01 IN-FILE-STATUS-CODE.

05 STATUS-KEY-ONE

PICTURE IS X.

05 STATUS-KEY-TWO

PICTURE IS X.

После выполнения ввода или вывода значением ключа состояния 1 может быть один из следующих символов:

0 — означает успешное завершение;

1 — означает конец файла;

2 — означает ошибку в ключе (используется только при относительной и индексной организации);

3 — означает постоянную ошибку, такую, как повреждение катушки с лентой или потеря разрядов данных.

Значением второго символа кода результата, ключа состояния 2, также является цифра; смысл этого символа прояснится после того, как в гл. 10 будет введено понятие ключевого значения. Второй сим-

вол обеспечивает дополнительную информацию о процессе обработки файла.

Фраза FILE STATUS IS представляет еще один пример средств КОБОЛа, обеспечивающих контроль правильности процесса обработки и гарантирующих выявление ошибок и отклонений. Обычно используются методы проверок, включающие вычисление контрольной цифры для определенных числовых данных наряду с рассмотренным использованием стандартных записей меток и ключей состояния файла. В рабочих применениях около 90% всех статей и операторов в отдельных случаях используется в целях контроля и проверок и только 10% — для задания самой обработки. В тексте данной книги не целесообразно постоянно уделять внимание повторяющимся проверкам правильности процесса обработки. Однако в реальных программах, предназначенных для выполнения процесса обработки данных в рабочих условиях, необходимо сделать все возможное для предотвращения отказов в системе и защиты заказчика от их последствий.

6.4. Параграф управления вводом-выводом

Только что описанные фразы были дополнительными фразами в параграфе FILE-CONTROL. Абсолютно новым параграфом в секции INPUT-OUTPUT SECTION (дополнительно к параграфу FILE-CONTROL и следуя за ним по порядку) является параграф I-O-CONTROL (УПРАВЛЕНИЕ-ВВОДОМ-ВЫВОДОМ). В параграфе I-O-CONTROL имеются следующие статьи:

I-O-CONTROL.

```
[ : SAME [RECORD] AREA FOR имя-файла-1 [имя-файла-2] ... ] ...
[ : MULTIPLE FILE TAPE CONTAINS имя-файла-3 [POSITION целое-1]
  [имя-файла-4 [POSITION целое-2]] ... ] ...
```

Приведем пример такого параграфа:

I-O-CONTROL.

```
SAME AREA FOR INPUT-FILE CARD-FILE READ-FILE
SAME RECORD AREA FOR PRINT-FILE OUT-FILE
MULTIPLE FILE TAPE CONTAINS
  REF-FILE-A POSITION 5
  STORAGE-FILE POSITION 23.
```

Параграф I-O-CONTROL не является обязательным как и каждая статья, которая может в нем появиться. Существуют две различные формы статьи SAME (ОБЩАЯ): SAME AREA (ОБЩАЯ ОБЛАСТЬ) и SAME RECORD AREA (ОБЩАЯ ОБЛАСТЬ ЗАПИСИ),

которые по смыслу отличаются друг от друга. Фраза SAME AREA определяет, что два или больше файлов в ходе выполнения операций ввода-вывода будут использовать одну и ту же область памяти, которая включает буфер, чередующиеся буферы и область записи из файловой области. Вследствие разделения этой области несколькими файлами одновременно может быть открыт только один файл из числа указанных. Таким образом, фраза

SAME AREA FOR INPUT-FILE CARD-FILE READ-FILE

означает, что в любой момент времени только один из этих трех файлов может быть открыт, но этот файл должен быть закрыт, прежде чем можно будет открыть другой файл. Если файлы можно обрабатывать по такому принципу, то с помощью этой фразы можно значительно сэкономить внутреннюю память.

Фраза SAME RECORD AREA также используется для экономии памяти, но в этом случае несколькими записями разделяется область записи из файловой области (остальная память, занятая под буферы, не разделяется). Все файлы, перечисленные во фразе, могут быть открыты в одно и то же время. Однако одновременно в области записи может находиться только одна запись данных, поэтому любой оператор READ или WRITE разрушает предыдущее содержимое области записи.

В параграфе I-O-CONTROL могут присутствовать несколько фраз SAME. Одно и то же имя-файла не может быть перечислено более чем в одной фразе SAME AREA или фразе SAME-RECORD AREA. Однако одно и то же имя-файла может быть перечислено и во фразе SAME AREA, и одновременно во фразе SAME RECORD AREA, но тогда в последней фразе должны быть перечислены все остальные имена-файлов из фразы SAME AREA. Кроме того, в одной фразе SAME RECORD AREA могут быть перечислены несколько групп имен-файлов из фраз SAME AREA, например как в приведенном ниже примере, в котором имена-файлов сокращенно обозначаются только буквами:

SAME AREA FOR A B C

SAME AREA FOR D E F G

SAME RECORD AREA FOR A B C D E F G

Одновременно могут быть открыты только один файл из группы A B C и только один файл из группы D E F G.

Фраза MULTIPLE FILE TAPE (НА ОДНОЙ КАТУШКЕ) может использоваться только в том случае, если несколько файлов располагаются на одной и той же физической катушке с лентой. Назначение этой фразы противоположно фразе FOR MULTIPLE REEL (НА НЕСКОЛЬКИХ КАТУШКАХ), которая указывает, что файл слишком большой и для его размещения требуется не-

сколько катушек. Во фразе MULTIPLE FILE TAPE подразумевается, что файлы достаточно малы и могут разместиться на одной катушке. Еще раз подчеркнем, что фраза MULTIPLE FILE TAPE применима только к катушкам с лентами. Заметим, что из числа файлов, находившихся на катушке, в этой фразе следует перечислять только те имена-файлов, которые используются в КОБОЛ-программе. Целое в варианте POSITION (ПОЗИЦИЯ) определяет местоположение каждого файла относительно начала ленты. Если имена-файлов перечислены в той последовательности, в которой они находятся на катушке, начиная с начала ленты, то использовать указание POSITION не имеет смысла. В любой момент времени может быть открыт только один из перечисленных файлов.

Ниже приводится программа, в которой используются возможности секции INPUT-OUTPUT SECTION. В программе обрабатываются три файла:

1. C-FILE, который хранится на магнитной ленте и по своим размерам занимает не целую катушку;
2. G-FILE, который хранится на той же катушке с лентой, что и C-FILE;
3. OUT-FILE, который требуется записать на одну, а если требуется, на несколько катушек лент.

Программа объединяет файлы C-FILE и G-FILE в новый файл OUT-FILE. Для экономии памяти для обоих входных файлов отводится общая область, и все три файла разделяют общую область записи. Файл C-FILE должен быть закрыт, чтобы можно было открыть файл G-FILE, но файл OUT-FILE должен быть открытым, когда открыт один из входных файлов.

IDENTIFICATION DIVISION.
PROGRAM-ID. PROG630.

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.
SOURCE-COMPUTER. COMPUTER-NAME.
OBJECT-COMPUTER. COMPUTER-NAME.

INPUT-OUTPUT SECTION.
FILE-CONTROL.

SELECT OUT-FILE
ASSIGN TO MAGNETIC-TAPE
RESERVE 4 AREAS.

SELECT C-FILE
 ASSIGN TO MAGNETIC-TAPE
 RESERVE 2 AREAS.

SELECT G-FILE
 ASSIGN TO MAGNETIC-TAPE
 RESERVE 2 AREAS.

I-O-CONTROL.

 SAME AREA FOR C-FILE G-FILE
 SAME RECORD AREA FOR OUT-FILE C-FILE G-FILE
 MULTIPLE FILE TAPE CONTAINS.
 C-FILE POSITION 3
 G-FILE POSITION 7.

DATA DIVISION.

FILE SECTION.

FD C-FILE

 LABEL RECORDS ARE STANDARD
 VALUE OF ID IS "INPUT"
 DATA RECORD IS INPUT-RECORD
 RECORD CONTAINS 20 CHARACTERS
 BLOCK CONTAINS 200 CHARACTERS.

01 INPUT-RECORD.

 05 FIELD-A PICTURE IS X(10).

 05 FIELD-B PICTURE IS X(10).

FD G-FILE

 LABEL RECORDS ARE STANDARD
 DATA RECORDS ARE G-RECORD-ONE
 G-RECORD-TWO
 RECORD CONTAINS 50 CHARACTERS
 BLOCK CONTAINS 200 CHARACTERS.

01 G-RECORD-ONE.

 05 RECORD-ONE-ID PICTURE IS X(5).

 05 RECORD-ONE-X PICTURE IS X(30).

 05 RECORD-ONE-Y PICTURE IS X(15).

01 G-RECORD-TWO.

 05 RECORD-TWO-ID PICTURE IS X(5).

 05 RECORD-TWO-A PICTURE IS X(45).

FD OUT-FILE

LABEL RECORDS ARE STANDARD

DATA RECORDS ARE SPECIAL-RECORD-A

SPECIAL-RECORD-B

RECORD CONTAINS 20 TO 50 CHARACTERS

BLOCK CONTAINS 1 RECORDS.

01 SPECIAL-RECORD-A PICTURE IS X(20).

01 SPECIAL-RECORD-B PICTURE IS X(50).

PROCEDURE DIVISION.

P-1.

OPEN INPUT C-FILE.

OPEN OUTPUT OUT-FILE.

P-2.

READ C-FILE RECORD AT END GO TO P-3.

WRITE SPECIAL-RECORD-A.

GO TO P-2.

P-3.

CLOSE C-FILE.

OPEN INPUT G-FILE.

READ G-FILE RECORD AT END GO TO P-4.

WRITE SPECIAL-RECORD-B.

GO TO P-3.

P-4.

CLOSE G-FILE OUT-FILE.

STOP RUN.

Упражнения

1. Нарисуйте схему распределения файловой памяти для описанного выше примера задачи. Изобразите расположение в памяти записей, буферных областей и общие области. Определите процесс перемещения данных, описанный в разделе процедур.

2. Четыре файла, каждый из которых сблокирован в блоки по 2500 символов, являются такими большими, что занимают целиком десять катушек. Файлы должны быть открыты и считываться одновременно. Завершив обработку, файлы необходимо закрыть и счи-

тать пятый файл (физическая запись которого содержит шестнадцать 200-символьных записей данных). Определенное числовое данное из каждой записи пятого файла требуется сравнить с некоторым числом, вычисленным в ходе первой обработки, и выбранные записи скопировать на шестой файл. Физическая запись этого файла также должна содержать шестнадцать 200-символьных записей. Изобразите параграфы управление-файлами и управление вводом-выводом так, чтобы в данной программе получилось экономное использование памяти.

3. Файл-А содержит записи длиной в пятьдесят символов. Файл-В содержит записи длиной в семьдесят-пять символов. Файл-С содержит записи длиной в 125 символов. Эти файлы, любой из которых может быть необязательным, должны обрабатываться последовательно с целью создания четвертого файла, выходного файла-D, состоящего из выбранных записей входных файлов. Все три входных файла размещаются на одной катушке с магнитной лентой. Напишите параграф управление-файлами, параграф управление-вводом-выводом и четыре статьи описания файлов, чтобы получить экономное распределение памяти для файлов.

6.5. Работа с файлом

Понятие записей разных форматов было введено наряду с фразой DATA RECORDS ARE из статьи-описания-файла. Записи данных могут иметь различные составы данных и различные длины. Все они независимо от длины считываются в одну и ту же область записи, расположенную в файловой области, и затем обычно перемещаются в рабочие области для дальнейших вычислений. Однако если все записи данных имеют одинаковую длину, хотя быть может и разные форматы, то следующий расширенный оператор READ позволяет совместить чтение и перемещение в рабочую область в одном операторе:

READ имя-файла RECORD [INTO идентификатор]; AT END повелительный-оператор

Когда используется вариант INTO (B), действие оператора эквивалентно следующим двум операторам:

READ имя-файла RECORD AT END повелительный-оператор.
MOVE имя-записи TO идентификатор.

где имя-записи — это имя любой из записей в файле. Какое именно имя-записи используется в операторе, не имеет значения, так как вариант INTO допустим только для записей одинакового размера, однако перемещение записи выполняется согласно правилам оператора MOVE. Идентификатор в варианте INTO должен быть либо

именем какой-либо статьи из секции рабочей памяти, либо именем записи из файловой области, соответствующей открытому в текущий момент выходному файлу. Оператор READ ... INTO подобен оператору WRITE ... FROM. Приведем примеры расширенного оператора READ:

```
READ INPUT-FILE INTO STORAGE-RECORD AT END GO  
TO P-5.
```

```
READ C-FILE RECORD INTO PRINT-LINE AT END MOVE  
"END" TO FLAG-X.
```

Когда при выполнении оператора READ произойдет исчерпывание всех записей данных файла на катушке, операционная система автоматически произведет смену катушки или диска, если файлу было назначено несколько катушек или дисков. Текущая катушка будет перемотана обратно (диск, конечно, не может быть перемотан), и оператору будет выдано сообщение о ее снятии. Среди физических устройств, назначенных файлу, система отыщет следующую катушку или диск. Если таковых не находится, то оператору будет выдано сообщение с требованием установить очередную катушку или диск. Если файл специфицирован фразой LABELS ARE STANDARD, то сначала будет выполнена стандартная процедура обработки начальной метки, после чего первая запись данных из новой катушки или диска будет помещена в область записи.

Когда считывается последняя лента или диск, наступает конец файла и выполняется повелительный оператор, стоящий за фразой AT END. После этого управление передается следующему оператору КОБОЛ-программы. При этом лента не перематывается. Для перемотки ленты требуется выполнить оператор CLOSE.

Программист должен быть готов к различным ситуациям, которые могут произойти при работе с файлом. Существуют файлы, для которых процесс обработки является простым и однонаправленным, например файлы, находящиеся на устройствах чтения перфокарт, перфораторах и устройствах печати. Файл для устройства чтения перфокарт может быть только входным, и чтение этого файла может выполняться только в одном направлении. Файлы для печатающих устройств могут быть определены только как выходные, и их запись производится только последовательно. Файлы на магнитных лентах являются более гибкими, так как их можно устанавливать в начальные положения за счет высокоскоростной перемотки лент и, кроме того, их можно читать в обратном направлении. Последняя особенность позволяет прочесть файл от начала до конца и затем считывать записи от конца к началу без перемотки ленты. При чтении ленты в обратном направлении записи помещаются в область записи в соответствии с форматом, определенным в описании (а не задом наперед), так как по сравнению с прямым чтением последовательность

доступа к записям файла меняется только с прямой на обратную. Для файлов, которым назначено несколько катушек, чтение в обратном направлении бессмысленно и поэтому не разрешается. Наиболее гибкими являются файлы на дисковом пакете, который исключительно удобен для хранения нескольких одновременно открытых файлов. (Фраза **MULTIPLE FILE TAPE CONTAINS** была определена только для катушки с лентой.) Однако тем не менее в КОБОЛе нет никаких команд для возврата к считанным областям. При последовательной обработке файла считанная запись может быть считана вновь, только если после закрытия файла снова открыть его и просмотреть с самого начала, считывая все записи вплоть до нужного места.

Для файлов, которые располагаются на одной катушке, в операторах **OPEN** и **CLOSE** используются дополнительные варианты, которые применяются либо для установки ленты в конец файлов, либо для последующего чтения в обратном направлении, либо для записи другого файла на эту же ленту. Если операторы **OPEN** или **CLOSE** используются без дополнительных вариантов, то лента полностью перематывается в начало, однако полный формат оператора **OPEN** вместе с дополнительными возможностями имеет вид:

$$\text{OPEN} \left\{ \begin{array}{l} \text{INPUT} \left\{ \begin{array}{l} \text{имя-файла-1} \\ \text{имя-файла-2} \\ \text{имя-файла-3} \end{array} \right\} \dots \\ \text{OUTPUT} \left\{ \begin{array}{l} \text{имя-файла-2} \\ \text{имя-файла-3} \end{array} \right\} \dots \\ \text{EXTEND} \left\{ \begin{array}{l} \text{имя-файла-3} \end{array} \right\} \dots \end{array} \right\} \left\{ \begin{array}{l} \text{REVERSED} \\ \text{WITH NO REWIND} \\ \text{WITH NO REWIND} \end{array} \right\} \dots \left\{ \dots \right\}$$

а формат оператора **CLOSE**, используемый только для файла, который располагается на одной катушке, имеет вид:

$$\text{CLOSE} \left\{ \text{имя-файла} \left[\text{WITH NO REWIND} \right] \right\} \dots$$

Приведем примеры употребления этих операторов:

OPEN INPUT CARD-FILE.

OPEN OUTPUT TAPE-OUT-FILE WITH NO REWIND.

OPEN INPUT MASTER-FILE REVERSED

SECOND-FILE WITH NO REWIND

THIRD-FILE.

OPEN EXTEND OLD-MASTER-FILE.

CLOSE FILE-A FILE-B FILE-C WITH NO REWIND.

В последнем операторе только файл **FILE-C** закрывается без перемотки, остальные закрываются стандартным образом, т. е. с перемоткой в начало. Файл можно закрыть стандартно на любом этапе обработки файла, но оператор **CLOSE WITH NO REWIND**

(ЗАКРЫТЬ БЕЗ ПЕРЕМОТКИ) следует употреблять только, когда наступает конец файла. Для разделения фраз в операторе могут использоваться запятые, которые необязательны. Варианты INPUT и OUTPUT уже упоминались раньше, но вариант EXTEND (ДОПОЛНЯЕМЫЙ) рассматривается впервые. В случае EXTEND файл при открытии устанавливается в положение после последней его записи. Последующие операторы WRITE будут добавлять новые записи к этому файлу, как если бы он был открыт с использованием варианта OUTPUT. Фраза EXTEND предназначена для расширения уже существующего файла путем добавления в него записей. Она может использоваться только для файлов, которые располагаются на одной катушке.

Для файлов, которые располагаются на нескольких катушках с лентами или дисках, существует свой вариант оператора CLOSE,

$$\text{CLOSE} \left\{ \text{ИМЯ-файла} \left[\left\{ \begin{array}{c} \text{REEL} \\ \text{UNIT} \end{array} \right\} \left\{ \begin{array}{c} \text{WITH NO REWIND} \\ \text{FOR REMOVAL} \end{array} \right\} \right] \right\} \dots$$

в котором фразы могут разделяться запятыми. Примерами такого оператора являются:

CLOSE FILE-A REEL WITH NO REWIND.
CLOSE FILE-B REEL FOR REMOVAL
FILE-C FILE-D UNIT FOR REMOVAL.

И опять необязательные слова в операторах относятся только к непосредственно предшествующим именам-файлов. В приведенном примере файл FILE-C закрывается окончательно и стандартным образом, а действие фразы UNIT FOR REMOVAL (ТОМ ¹⁾ С УДАЛЕНИЕМ) распространяется только на файл FILE-D. Ниже приводятся примеры употребления этого оператора:

CLOSE FILE-X REEL FILE-Y UNIT FILE-Z.
CLOSE FILE-W REEL FOR REMOVAL FILE-V.

Вариант REEL/UNIT (КАТУШКА/ТОМ) совершенно не допустим для файлов, располагаемых либо на одной катушке, либо на одном диске, и если программа попытается выполнить оператор CLOSE REEL для файла на одной катушке или дисковом томе, то результаты будут непредсказуемы. Вариант REEL (КАТУШКА) или UNIT (ТОМ) закрывает обрабатываемую в данный момент катушку или том и в случае ленты перематывает ее, если не был определен

¹⁾ Вариант ТОМ используется для файлов, расположенных на дисках.—
Прим. перев.

вариант NO REWIND. После этого достается следующая катушка (том), на которой располагается файл. Для этого оператору посылается сообщение с требованием о снятии текущей катушки (тома) и установке следующей катушки (тома). Варианты CLOSE UNIT и CLOSE REEL используются для ускорения поиска требуемой записи в файле, располагаемом на нескольких катушках или томах, когда становится очевидным, что этой записи нет на текущей катушке или томе.

Фраза FOR REMOVAL (С УДАЛЕНИЕМ) используется для сообщения оператору машины, что данная катушка или том удаляются из текущей обработки. Чаще всего, но не всегда, такое сообщение позволяет оператору физически снять катушку или том с соответствующего устройства, которое может быть использовано для других целей. Однако вариант FOR REMOVAL относится к текущей катушке или тому, а не ко всему файлу. Окончательно закрыть файл можно, только употребив оператор CLOSE FILE-W без вариантов REEL/UNIT. Таким образом, от оператора машины можно потребовать снятия конкретной катушки, задав следующую последовательность операторов (не обязательно следующих подряд, но расположенных в указанном порядке):

CLOSE FILE-W REEL FOR REMOVAL.

CLOSE FILE-W.

OPEN INPUT FILE-W.

Когда лента потребуется снова, оператору будет послано соответствующее сообщение.

Последний вариант для оператора CLOSE записывается так:

CLOSE {имя-файла [WITH LOCK]}

В этом случае файл закрывается и запирается, предотвращая дальнейшее его использование в процессе работы программы. После такого закрытия файл нельзя «отпереть» никакими способами. Этот оператор безопасности используется, чтобы защитить файл от попыток открыть его как выходной и внести в него какие-нибудь изменения. Оператор CLOSE ... WITH LOCK (ЗАКРЫТЬ ... С ЗАМКОМ) может использоваться для файлов любого типа. Полное описание оператора CLOSE имеет вид:

$$\underline{\text{CLOSE}} \left\{ \begin{array}{l} \text{имя-файла} \left[\begin{array}{l} \left\{ \begin{array}{l} \underline{\text{REEL}} \\ \underline{\text{UNIT}} \end{array} \right\} \left\{ \begin{array}{l} \text{WITH NO REWIND} \\ \text{FOR REMOVAL} \end{array} \right\} \\ \text{WITH} \left\{ \begin{array}{l} \underline{\text{NO REWIND}} \\ \underline{\text{LOCK}} \end{array} \right\} \end{array} \right] \end{array} \right\}.$$

Упражнения

1. Заполните пропуски (иногда более чем одним словом):

а) Для конкретного файла оператор _____ должен выполняться раньше операторов READ или WRITE.

б) Операторы OPEN и _____ всегда должны быть попарно связаны.

в) Два варианта _____ и _____ могут использоваться только для последовательного файла, располагаемого на одной катушке или диске.

г) Когда файл открывается с использованием варианта WITH NO REWIND, то он уже должен быть установлен в _____; когда файл _____ с использованием варианта WITH NO REWIND, то он может быть установлен в любом месте.

д) Вариант CLOSE _____ недопустим для файла с последовательным доступом, который располагается на одной катушке.

е) Если файл нельзя повторно обработать за время выполнения текущей программы, то говорят, что он закрыт _____.

ж) Когда используется простой оператор _____ без всяких вариантов, то определяемый им файл устанавливается в начало.

2. Напишите только раздел процедур для копирования одного файла (располагаемого на одной ленте) на другой файл и затем, запись за записью, сравните их между собой, чтобы убедиться, что при переписывании не произошло ошибок. Сэкономьте время проверки, не перематывая ленты после копирования.

3. Напишите полную КОБОЛ-программу, обращая внимание на статьи FD и параграфы раздела оборудования, которая выполняет следующую работу:

а) Сликает два входных файла в один выходной файл, копируя все записи первого файла на выходной файл, а затем продолжает копирование записей второго файла, не закрывая выходной файл.

б) Два входных файла описаны без использования варианта BLOCK (каждая физическая запись содержит ровно одну логическую запись), и все записи состоят ровно из двадцати символов. Каждая физическая запись выходного файла должна содержать сто записей.

в) После создания объединенного файла скопируйте его четыре раза на четыре файла, располагаемые на одной и той же катушке с лентой.

4. Имеется единственный входной файл, располагаемый на нескольких катушках с лентами. В пределах каждой катушки записи файла упорядочены по идентификационному-номеру-покупателя, но это не справедливо для всего файла. Считайте одну запись из другого файла, содержащую определенный идентификационный-номер-покупателя. Последовательно просматривайте основной файл до совпадения записей или до исчерпывания файла. Как только

идентификационный-номер-покупателя на данной катушке оказывается больше заданного числа, закройте данную катушку и продолжайте поиск на следующей катушке. КОБОЛ-программу для этого примера напишите полностью.

6.6. Декларативные секции

Всегда существуют процедуры, которые требуется выполнять в разных местах программы, и желательно не описывать их по многу раз. В КОБОЛе предусмотрены определенные возможности для организации передач управления в изолированные секции программ. После выполнения секции управление автоматически возвращается в ту точку программы, откуда произошла передача управления.

Эти изолированные секции, выполняющиеся только в особых случаях, таких, как ошибка при выполнении входной операции, называются *декларативными секциями* и группируются все вместе в самом начале раздела процедур. Декларативные секции изолированы от остальных процедур ключевым словом **DECLARATIVES** (ДЕКЛАРАТИВЫ), предшествующим этим секциям, и завершаются словами **END DECLARATIVES** (КОНЕЦ ДЕКЛАРАТИВ). Причина такого разделения и изоляции заключается в том, что декларативные секции получают управление только при некоторых определенных условиях и не могут выполняться в обычной последовательности команд. Например, декларативная секция может быть определена как выполняемая только при возникновении ошибки в передаче данных в момент чтения какого-нибудь входного файла. Секции, заключенные между обрамляющими их словами **DECLARATIVES** и **END DECLARATIVES**, выполняются только тогда, когда возникают те или иные определенные условия. Если в программе используются декларативные секции, то выполнение процедур начинается с первого параграфа, следующего за словами **END DECLARATIVES**, и продолжается до тех пор, пока не возникнут особые условия. В случае такого условия управление передается в декларативную секцию. После выполнения последнего оператора из этой секции, управление возвращается в основную программу (если в декларативной секции не был выполнен оператор **STOP RUN**).

Теперь описание раздела процедур требуется расширить следующим образом:

PROCEDURE DIVISION.
DECLARATIVES.

Имя-секции SECTION.

Декларативное-предложение.

Имя-параграфа.

Предложение.

END DECLARATIVES.

Имя-секции SECTION.

Имя-параграфа.

Предложение.

Для упрощения здесь опущены квадратные и фигурные скобки, необходимые для указания повторяемости предложений и параграфов. Внутри параграфов предложения могут повторяться, а параграфы могут повторяться внутри секций. В обеих частях раздела процедур может быть много секций. Декларативная часть должна быть представлена в виде секций, а когда хотя бы один параграф в программе заключен в секцию, то все остальные параграфы также должны быть заключены в секции. Например, так:

PROCEDURE DIVISION.

DECLARATIVES.

INPUT-ERROR-NOTIFICATION SECTION.

USE AFTER STANDARD ERROR PROCEDURE
ON CARD-IMAGE-FILE.

PARAGRAPH-ONE.

DISPLAY

"A NONRECOVERABLE INPUT ERROR HAS
OCCURRED"

"ON CARD-IMAGE-FILE"

UPON CONSOLE.

END DECLARATIVES.

INITIALIZATION SECTION.

P-1.

OPEN INPUT CARD-IMAGE-FILE.

READ CARD-IMAGE-FILE RECORD AT END

GO TO P-10.

P-2.

.
.
.

Выполнение этой программы начнется с первого оператора в параграфе P-1, и секция INPUT-ERROR-NOTIFICATION не будет выполняться до тех пор, пока не возникнет ошибка в операторе типа READ CARD-IMAGE-FILE... . Тогда будет выполнен оператор DISP-

LAY из параграфа PARAGRAPH-ONE и управление вернется к параграфу P-2.

В КОБОЛе имеется несколько форм допустимых декларативных предложений, например предложение USE FOR DEBUGGING (ИСПОЛЬЗОВАТЬ ДЛЯ ОТЛАДКИ), которое будет описано в гл. 11. Сейчас будет описана только одна форма декларативных предложений, которая используется в случае возникновения ошибок:

<u>USE AFTER STANDARD</u>	$\left. \begin{array}{c} \text{EXCEPTION} \\ \text{ERROR} \end{array} \right\}$	<u>PROCEDURE ON</u>
$\left\{ \begin{array}{l} \text{имя-файла-1} \text{ [имя-файла-2] } \dots \\ \text{INPUT} \\ \text{OUTPUT} \\ \text{EXTEND} \end{array} \right\}$		

В предложении возможны различные комбинации вариантов, например:

```
USE AFTER STANDARD EXCEPTION PROCEDURE
    ON CARD-IMAGE-FILE OUT-FILE-ZULU.
USE AFTER STANDARD ERROR PROCEDURE ON INPUT.
USE AFTER STANDARD EXCEPTION PROCEDURE ON
    EXTEND.
```

Декларативное предложение определяет ситуацию, при которой управление передается в данную декларативную секцию. Декларативное предложение не выполняется — это описательный оператор. В программе может быть много декларативных секций и в каждой из них может быть только одно декларативное предложение.

В данном контексте слова EXCEPTION (ИСКЛЮЧЕНИЕ) и ERROR (ОШИБКА) являются синонимами, поэтому фактически в предложении USE AFTER STANDARD ERROR (ИСПОЛЬЗОВАТЬ ПОСЛЕ СТАНДАРТНОЙ ОШИБКИ) имеется только одна альтернатива: выбор одного из вариантов INPUT, OUTPUT, EXTEND или имени конкретного файла (имен конкретных файлов), используемого в программе. Секция, описываемая предложением USE (ИСПОЛЬЗОВАТЬ), может быть активирована только в результате выполнения в основной программе операторов READ или WRITE. Ключевые слова определяют, какой из операторов, READ или WRITE, может инициировать вызов декларативной секции. Если используется ключевое слово INPUT, то все файлы, описанные как INPUT, являются потенциальными источниками инициирования этой декларативной секции при выполнении оператора READ.

Ключевые слова OUTPUT и EXTEND идентичны. Для всех файлов, открытых с аналогичными вариантами, оператор WRITE может вызывать передачу управления в декларативные секции. Ключевые слова взаимно исключаемы, и в конкретном декларативном предложении может использоваться только одно из них. Таким образом, если программист желает охватить все файлы, то ему необходимо иметь несколько декларативных секций. Использование конкретных имен-файлов ограничивает применение секции, распространяясь только на перечисленные файлы. Однако в этом случае передача управления может инициироваться как операторами READ, так и WRITE. Программист не имеет возможности ограничивать описание декларативного предложения таким образом, чтобы его действие распространялось только либо на чтение, либо на запись указанного файла. Однако на этой стадии описания КОБОЛа это не вызывает затруднений, потому что файлы с последовательной организацией и последовательным доступом (единственные до сих пор рассмотренные способы) могут быть открыты только как входные, выходные или дополняемые, но никогда как входные и выходные одновременно. В гл. 10 будет описана другая организация файла, располагаемого на дисковых устройствах, при которой файл может быть открыт как входной и выходной одновременно (I-O). Для таких файлов декларативное предложение может иметь такой вид:

USE AFTER STANDARD ERROR PROCEDURE ON I-O.
(ИСПОЛЬЗОВАТЬ ПОСЛЕ СТАНДАРТНОЙ ПРОЦЕДУРЫ
ОШИБКИ ДЛЯ ВХОДНЫХ-ВЫХОДНЫХ)

Операторы READ или WRITE могут активировать декларативную секцию, описываемую конкретным предложением USE, по двум причинам. Первая из них — неустраняемая ошибка во входной или выходной операции, которая может произойти, когда машина пытается передать блок или запись из внутренней памяти на внешнее устройство, назначенное файлу, или наоборот. Такие ошибки могут возникнуть при действительном повреждении физической поверхности катушки или диска или от загрязнения записывающей поверхности. На вычислительных установках стараются не допускать таких ошибок поддержанием высокого уровня чистоты, требуемой температуры и влажности воздуха и профилактическими осмотрами внешних устройств. Кроме физических повреждений в вычислительной системе случаются другие неисправности — уменьшение напряжения, вызываемое падением напряжения в сети и помехами, или выход из строя некоторых электронных компонентов. В таких случаях возникают неустраняемые ошибки, и обычно единственное, что может предпринять программист в декларативной секции, это закрыть файл и выдать об этом сообщение по оператору DISPLAY. Если, несмотря на ошибку, обработка может

быть продолжена, то управление возвращается к оператору, следующему за оператором READ или WRITE, и нормальная последовательность вычислений продолжается.

Вторая причина, по которой может быть вызвана декларативная секция при выполнении операторов READ или WRITE — это попытка передать запись вне границ файла. Для чисто последовательных файлов это происходит только в том случае, если оператор пытается получить доступ к записи после исчерпывания файла. Такая ситуация обычно отслеживалась фразой AT END из предложения READ, например так:

READ INPUT-FILE RECORD AT END GO TO P-10.

Однако если в декларативной части программы имеется предложение USE, то фразу AT END в операторе READ, относящемуся к этому файлу, можно не употреблять. Таким образом, если предложение USE имеет вид

USE AFTER STANDARD ERROR PROCEDURE ON
INPUT-FILE.

и записывается после имени-секции, заключенной между словами DECLARATIVES и END DECLARATIVES, то возможно использование простого оператора-READ:

READ INPUT-FILE RECORD.

Фраза INTO, как и раньше, является необязательной. Поэтому если все записи в файле одинакового размера, то возможен и такой оператор:

READ INPUT-FILE RECORD INTO SOME-OTHER-RECORD.

Программист имеет возможность определить, по какой причине произошла активация декларативной секции, проверив значение данного, именуемого статьей

FILE STATUS IS имя-данного

т.е. служащего для помещения кода результата операции ввода-вывода. Значение ключа состояния 1, первого символа из этого двухсимвольного данного, принимает значение 1 в случае конца файла и 3 — в случае неустранимой ошибки.

Ключ состояния 1 принимает значение 2 в тех случаях, когда происходит ошибка в ключе записи. Такая ситуация возникает также при попытке использования файла за его границами, но возможна только для файлов с *индексной* или *относительной* организацией (рассмотрение которых отложено до гл. 10).

Другим ограничением для декларативных секций является запрещение ссылок из них к операторам, описанным вне декларатив-

IDENTIFICATION DIVISION
PROGRAM-ID. PROG650.

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER. COMPUTER-NAME.
OBJECT-COMPUTER. COMPUTER-NAME.
SPECIAL-NAMES.

CONSOLE IS OPERATOR-TYPEWRITER.

- * СЛОВО CONSOLE ОПРЕДЕЛЯЕТСЯ В РЕАЛИЗАЦИИ. СЛОВО
- * OPERATOR-TYPEWRITER СОЗДАЕТСЯ ПРОГРАММИСТОМ.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

SELECT INPUT-FILE

ASSIGN TO MAGNETIC-TAPE

RESERVE 3 AREAS

ORGANIZATION IS SEQUENTIAL

ACCESS MODE IS SEQUENTIAL

FILE STATUS IS FILE-STATUS-VALUE.

SELECT OUTPUT-FILE

ASSIGN TO MAGNETIC-TAPE

RESERVE 1 AREA

ORGANIZATION IS SEQUENTIAL

ACCESS MODE IS SEQUENTIAL.

- * ФРАЗА FILE STATUS НЕОБЯЗАТЕЛЬНА.

I-O-CONTROL.

MULTIPLE FILE TAPE CONTAINS INPUT-FILE POSITION 4.

SAME RECORD AREA FOR INPUT-FILE OUTPUT-FILE.

DATA DIVISION.

FILE SECTION.

FD INPUT-FILE

LABEL RECORDS ARE STANDARD

VALUE OF IDENTIFICATION IS "IN593"

DATA RECORD IS IN-REC

RECORD CONTAINS 50 CHARACTERS

BLOCK CONTAINS 500 CHARACTERS.

01 IN-REC.

05 TEST-POSITION PICTURE IS X.

05 FILLER PICTURE IS X(49).

FD OUTPUT-FILE

LABEL RECORDS ARE STANDARD

VALUE OF IDENTIFICATION IS "OUT260"

DATA RECORD IS OUT-REC

RECORD CONTAINS 50 CHARACTERS

BLOCK CONTAINS 2500 CHARACTERS.

01 OUT-REC

PICTURE IS X(50).

- * ЗАПИСИ IN-REC И OUT-REC ИСПОЛЬЗУЮТ

Рис. 6.6. Пример программы, использующей оператор EXIT.

- ОДНУ И ТУ ЖЕ ОБЛАСТЬ ЗАПИСИ В ПАМЯТИ, НО РАЗНЫЕ
- ФАЙЛОВЫЕ БУФЕРЫ. ОБА ФАЙЛА МОГУТ
- БЫТЬ ОТКРЫТЫ ОДНОВРЕМЕННО.

WORKING-STORAGE SECTION.

01 FILE-STATUS-VALUE.

05 FIRST-KEY PICTURE IS X.

05 SECOND-KEY PICTURE IS X.

01 COUNT-OF-ERRORS PICTURE IS 99 VALUE IS ZERO.

PROCEDURE DIVISION.

DECLARATIVES.

EXCEPTION-ON-INPUT-FILE SECTION.

USE AFTER STANDARD ERROR PROCEDURE ON INPUT-FILE.

P-1.

IF FIRST-KEY IS EQUAL TO "1"

CLOSE INPUT-FILE OUTPUT-FILE

STOP RUN.

P-2.

MOVE SPACE TO TEST-POSITION.

ADD 1 TO COUNT-OF-ERRORS.

IF COUNT-OF-ERRORS IS GREATER THAN 20 DISPLAY

"NONRECOVERABLE ERROR" ON OPERATOR-TYPEWRITER CLOSE INPUT-FILE

OUTPUT-FILE

STOP RUN.

P-3.

EXIT.

EXCEPTION-ON-OUTPUT-FILE SECTION.

USE AFTER STANDARD ERROR PROCEDURE ON OUTPUT-FILE.

S-1.

DISPLAY "NONRECOVERABLE ERROR ON OUTPUT" ON OPERATOR-TYPEWRITER.

CLOSE INPUT-FILE OUTPUT-FILE.

STOP RUN.

END DECLARATIVES.

MAIN-PROGRAM SECTION.

M-1.

OPEN INPUT INPUT-FILE EXPAND OUTPUT-FILE.

M-2.

READ INPUT-FILE RECORD.

IF TEST-POSITION IS EQUAL TO "B" WRITE OUT-REC:

- ЗАМЕТИМ—ПРОГРАММА КОПИРУЕТ В ВЫХОДНОЙ-ФАЙЛ ТОЛЬКО ТЕ ЗАПИСИ
- ВХОДНОГО-ФАЙЛА, КОТОРЫЕ СОДЕРЖАТ „В“ В ПЕРВОЙ
- СИМВОЛЬНОЙ-ПОЗИЦИИ.

GO TO M-2.

Рис. 6.6. Пример программы, использующей оператор EXIT (продолжение).

ной секции. В обычных процедурных секциях также запрещено ссылаться к операторам декларативной секции. Таким образом, эти два класса секций должны быть изолированы друг от друга. Наиболее важным ограничением является запрещение возврата в основную программу до выполнения в декларативной секции последнего повелительного оператора. Когда в декларативной секции содержатся только повелительные операторы, то этих проблем не возникает, так как выполнение начинается с первого оператора этой секции и последовательно переходит с одного оператора на другой до тех пор, пока не будет выполнен последний повелительный оператор секции, после чего управление возвращается в основную программу. Когда в декларативной секции имеются условные операторы, то все возможные пути управления должны завершаться в самом конце секции. Запрещается возврат в основную программу из середины декларативной секции. Однако программист может и не располагать таким общим оператором, который мог бы завершить все возможные разветвления в программе. Конечно, можно специально для этого ввести какой-нибудь бесполезный оператор, например так:

PARAGRAPH-TEN.

MOVE ZERO TO PLACE-WHERE-ZERO-IS.

но при этом используемое данное требуется описать в статье-описания данного, и, кроме того, если через несколько месяцев программа будет кем-то пересматриваться, то сам оператор, т. е. засылка нуля в данное, может вызвать недоумение. Для такого случая в КОБОЛе предусмотрено фиктивное предложение. Она рассматривается как повелительное предложение, хотя фактически ничего не делает и играет роль общей завершающей точки для всех программных разветвлений, сходящихся в конце декларативной секции (или в некоторых других особых случаях, которые описываются в гл. 9). Предложение имеет вид

имя-параграфа.

EXIT.

Описание включает имя-параграфа, так как предложение EXIT (ВЫХОД) должно употребляться только в параграфе и должно быть единственным в нем. Например:

END-PARAGRAPH. EXIT.

В программе на рис. 6.6 показано использование оператора EXIT и остальных фраз, описанных в этой главе.

Глава 7. Структуризация данных

7.1. Статьи комментариев

Ранее был описан лишь один параграф раздела идентификации (IDENTIFICATION DIVISION), а именно: параграф PROGRAM-ID (ПРОГРАММА). Это единственный параграф данного раздела, который должен присутствовать в каждой исходной КОБОЛ-программе. Однако кроме него программист может включать в раздел идентификации параграфы, содержащие дополнительную информацию о программе, ее назначении и времени написания и компиляции программы. Хотя эти параграфы являются необязательными, рекомендуется включать их в каждую программу. Назначение программы заключается в использовании вычислительной машины для эффективного решения конкретной задачи, а не в ослаблении требований, предъявляемых программисту. Во время использования программы для решения реально возникающих задач ее требуется соответствующим образом обслуживать. Либо из-за ограничений на программу, либо, что даже чаще имеет место, в результате изменения в постановке задачи становится необходимым вносить изменения в программу. Такое обновление программы облегчается при наличии подробной документации, помогающей тому, кто вносит изменения в программу, ясно представить себе намерения ее автора. Даже в том случае, когда изменения вносятся самим автором программы по прошествии сравнительно небольшого отрезка времени после ее написания, такая информация оказывается полезной и даже необходимой для того, чтобы эти изменения были внесены правильно.

Общий формат раздела IDENTIFICATION DIVISION включает один обязательный и шесть необязательных параграфов. Имена этих параграфов фиксированы и представляют собой зарезервированные слова. При использовании каких-либо из этих параграфов их необходимо располагать в указанном ниже порядке:

IDENTIFICATION DIVISION.

PROGRAM-ID. имя-программы.

[AUTHOR. [статья-комментарий]]

[INSTALLATION. [статья-комментарий]]

[DATE-WRITTEN. [статья-комментарий]]

[DATE-COMPILED. [статья-комментарий]]

[SECURITY. [статья-комментарий]]

Статья-комментарий представляет собой строку литер из набора литер вычислительной машины, а не только из более ограниченного набора литер КОБОЛа. Постольку поскольку статья-комментарий располагается в поле В, для нее допустима любая комбинация литер, включая точки и зарезервированные слова. Таким образом, следующий пример раздела идентификации является правильным:

IDENTIFICATION DIVISION.

PROGRAM-ID. EXAMPLE.

AUTHOR. АВТОР ЭТОЙ ПРОГРАММЫ С ИМЕНЕМ EXAMPLE
РАБОТАЕТ НА ДАННОЙ УСТАНОВКЕ С ИЮНЯ
1972 Г.

INSTALLATION.

```
* * * * *
*
*   ОРИГИНАЛЬНЫЙ ПРИМЕР!
*
* * * * *
```

Правила написания статей-комментариев для различных параграфов абсолютно одинаковы. Имя автора может быть записано в параграфе SECURITY (ПОЛНОМОЧИЯ), не влияя на компиляцию. Статья-комментарий просто воспроизводится в распечатке программы в таком виде, в каком она была введена. Так как для записи статьи-комментария могут использоваться любые литеры, содержание параграфа DATE-WRITTEN (ДАТА-НАПИСАНИЯ) может быть представлено в любой удобной для программиста форме, например:

```
JANUARY 25, 1974
25 JAN 1974
7401025
01/25/74
```

Параграф DATE-COMPILED (ДАТА-ТРАНСЛЯЦИИ) отличается от других параграфов тем, что статья-комментарий, написанная в исходной программе, в процессе компиляции будет заменена на текущую дату. Таким образом, в распечатке программы указывается фактическая дата компиляции, так что программист всегда может определить, какая из распечаток является более новой. Следовательно, если бы программист написал такой параграф

DATE COMPILED. JUNE 15, 1974.

а на самом деле программа компилировалась бы в декабре, то в распечатке программы был бы напечатан, например, параграф

DATE-COMPILED. DEC 9, 1974.

Даты, указанные в параграфах DATE-WRITTEN и DATE-COMPILED, могут существенно различаться, в особенности если в программу вносилось много изменений. Для правильного обслуживания программы разумно вернуться к предыдущим программам и вставить дополнительные статьи-комментарии, указывающие, что программа все еще используется:

DATE-WRITTEN. APRIL 5, 1974.

НИКАКИХ ИЗМЕНЕНИЙ—ПРОГРАММА РАБОТАЕТ

В ОКТЯБРЕ 15, 1975.

DATE-COMPILED.

СЮДА ЗАНОСИТСЯ ТЕКУЩАЯ ДАТА.

Такие статьи-комментарии могут появляться только в разделе идентификации. Существует другая форма статьи-комментария, называемая строкой комментария, которая может присутствовать в любом из четырех разделов. Строка комментария выделяется с помощью звездочки (*) или дробной черты (/), помещаемых в поле индикатора строки (обычно в седьмую позицию строки). Такую строку можно поместить в любое место исходной КОБОЛ-программы. Единственное ограничение заключается в том, что она не может быть первой или последней строкой программы. Строка комментария распечатывается как часть исходной программы, но игнорируется при компиляции. Например, в параграфе

PARAGRAPH-FOUR.

MOVE ZERO TO ANSWER-X.

* ПРИСВОЕНИЕ НАЧАЛЬНОГО ЗНАЧЕНИЯ ДАННОМУ
* ANSWER-X.

WRITE ANSWER-RECORD.

содержится строка комментария. При компиляции эта строка комментария будет игнорироваться и программа будет скомпилирована так, как если бы этот параграф был записан в виде

PARAGRAPH-FOUR.

MOVE ZERO TO ANSWER-X.

WRITE ANSWER-RECORD.

или каким-либо другим способом.

Статьи-комментарии раздела идентификации могут содержать любую литеру из набора литер машины, но должны располагаться на бланке только в поле В. Строка комментария также может со-

держат любую литеру из набора литер машины и, кроме того, может быть записана как в поле А, так и в поле В. Использование признака строки комментария (* или /) в поле индикатора исключает возможность указания продолжения предыдущей строки с помощью дефиса (-). Однако в этом нет необходимости, так как каждая строка комментария все равно игнорируется и единый комментарий может быть записан в виде нескольких строк комментария, расположенных подряд. Например:

- * ДЛИННОЕ ОПИСАНИЕ МОЖНО ЗАДАТЬ С ПОМОЩЬЮ
- * НЕСКОЛЬКИХ СТРОК КОММЕНТАРИЯ,
- * ЗАПИСАННЫХ ПОСЛЕДОВАТЕЛЬНО.
- * ЗАРЕЗЕРВИРОВАННЫЕ СЛОВА КОБОЛА,
- * ТАК ЖЕ КАК И ЗНАКИ ПРЕПИНАНИЯ, В СТРОКАХ
- * КОММЕНТАРИЯ НЕ ВЛИЯЮТ НА ПРОЦЕСС
- * КОМПИЛЯЦИИ!

Различие между звездочкой и дробной чертой для обозначения строки комментария заключается в том, что при распечатке исходной программы строка комментария со звездочкой печатается в очередной строке, а строка комментария с дробной чертой печатается, начиная с новой страницы. Тем самым программист может управлять внешним видом распечатываемой исходной программы и начинать распечатку разделов или секций с новой страницы. Рекомендуется обязательно использовать строки комментариев, так как хорошо документированная программа проще для понимания и при необходимости в нее легче вносить изменения.

7.2. Дополнительные фразы описания данных

Статья-описания-записи и статья-описания-данного определяют организацию данных, хранящихся соответственно в файловой области и области рабочей-памяти внутренней памяти машины. Эти две статьи описаний имеют сходное назначение и почти одинаковый формат. Они различаются только в силу различных свойств области рабочей-памяти и файловой области. Записи, хранящиеся в файловой области, обычно существуют очень недолго, так как выполнение операторов READ и WRITE разрушает их значения. Записи, хранящиеся в области рабочей-памяти, могут обладать начальными значениями и используются для запоминания промежуточных результатов вычислений. Значения данных в рабочей-памяти изменяются только в том случае, когда они являются принимающими данными в операторах пересылки или арифметических операторах.

Ранее упоминалось лишь единственное различие между статьей-описания-записи и статьей-описания-данного, заключающееся в возможности использования фразы

VALUE IS литерал

которая допускается для статьи-описания-данного и запрещается для статьи-описания-записи. Фраза **VALUE IS** (ЗНАЧЕНИЕ) отличается от фразы **VALUE OF** (ЗНАЧЕНИЕ) статьи-описания-файла. Фраза **VALUE IS** может применяться как для элементарных, так и для групповых данных. При этом существуют определенные ограничения на использование этой фразы для групповых данных. Все данные, из которых составлено групповое данное, должны быть одного и того же размера. В качестве литералов можно использовать только нечисловые значения и стандартные константы. Ни для одного из подчиненных данных, составляющих групповое данное, нельзя дополнительно задавать собственную фразу **VALUE IS**. Для подчиненных данных не должны быть также заданы фразы **JUSTIFIED** (СДВИНУТО), **SINCHRONIZED** (ВЫДЕЛЕНО) или **USAGE** (ДЛЯ) (ни одна из этих фраз до сих пор не упоминалась). Пример группового данного с фразой **VALUE IS**:

WORKING-STORAGE SECTION.

01	GROUP-ITEM-X	VALUE IS SPACES.
05	DATA-ITEM-A	PICTURE IS X(5).
05	DATA-ITEM-B	PICTURE IS X(5).

Существует еще одно различие между статьей-описания-записи и статьей-описания-данного. Номер-уровня, который может принимать значения от 01 до 49 включительно (ведущие нули могут быть опущены, так что 01 и 1 означают один и тот же уровень), для статей в секции рабочей-памяти может принимать еще одно специальное значение 77. Номер-уровня 77 используется только для элементарных данных, которые предназначены для самостоятельного хранения в области рабочей-памяти, а не в составе какой-либо записи. Особых причин для использования статей-описания-данного уровня 77 нет, но на некоторых вычислительных машинах их использование может привести к некоторой экономии внутренней памяти, так как записи на этих машинах могут начинаться лишь в определенных ячейках памяти. За исключением этого, между следующими двумя статьями нет никакого различия:

WORKING-STORAGE SECTION.

77	STORAGE-PLACE-X	VALUE IS 35 PICTURE IS 9(3)V9.
01	STORAGE-PLACE-Y	VALUE IS 35 PICTURE IS 9(3)V9.

При использовании статьи уровня 77 номер уровня 77 должен начинаться на бланке в поле А точно так же, как и номер уровня 01.

Статьи уровня 77 могут быть написаны в любом месте секции рабочей-памяти и не обязательно должны предшествовать статьям уровня 01.

Между статьями-описания-записи и статьями-описания-данного, помимо использования фразы VALUE IS и статей уровня 77, нет никаких других различий. Все необязательные фразы, рассматриваемые ниже, с равным успехом относятся и к файловой области, и к области рабочей-памяти. В данном разделе описываются три таких фразы: фраза JUSTIFIED RIGHT (СДВИНУТО ВПРАВО), фраза BLANK WHEN ZERO (ПРОБЕЛ КОГДА НУЛЬ) и фраза SIGN (ЗНАК). Эти фразы представлены в следующем формате статьи-описания:

номер-уровня	$\left\{ \begin{array}{l} \text{имя-данного} \\ \text{FILLER} \end{array} \right\}$	
[фраза PICTURE]		(требуется элементарного уровня)
[фраза VALUE]		(может встречаться на групповом уровне)
[фраза JUSTIFIED]		(фразы JUSTIFIED и BLANK являются взаимоисключающими)
[фраза BLANK]		
[фраза SIGN]		(требуется символа S в шаблоне)

Эти фразы могут быть записаны в любом порядке.

Фраза JUSTIFIED такова:

$$\left\{ \begin{array}{l} \text{JUSTIFIED} \\ \text{JUST} \end{array} \right\} \text{ RIGHT}$$

Допускается сокращенная форма JUST, а необязательное слово RIGHT (ВПРАВО) используется для ясности. Фразу JUSTIFIED RIGHT нельзя задавать для числовых или для числовых-редактируемых данных. Она может использоваться для буквенных и буквенно-цифровых (отличных от числовых-редактируемых) данных. Эта фраза может присутствовать только на уровне элементарного данного. Когда одно из таких данных является принимающим при передаче данных (в операторе READ или MOVE) и в его описании присутствует фраза JUSTIFIED RIGHT, то пересылаемое данное размещается справа налево, начиная с правой границы принимающего данного в отличие от обычного расположения, при котором пересылаемое данное размещается слева направо, начиная с левой границы принимающего данного. Если пересылаемое данное длиннее принимающего данного, то его крайние левые литеры усекаются. Если пересылаемое данное короче, то на левом конце принимающего данного вставляется соответствующее число пробелов. Таким образом, стандартная передача данных, при которой литеры размещаются слева направо, для данных, описанных с помощью

фразы JUSTIFIED RIGHT, превращается в передачу, при которой литеры размещаются справа налево.

Фраза BLANK имеет следующий вид:

BLANK WHEN ZERO

Для ясности предлагается использовать эту фразу целиком, т. е. вместе с необязательным словом WHEN. Фраза BLANK WHEN ZERO может быть использована только для элементарного данного с фразой PICTURE (ШАБЛОН), определяющей числовую или числовую редактируемую категорию. В силу этого фразы JUSTIFIED RIGHT и BLANK WHEN ZERO никогда не могут встретиться в описании одного данного. Когда фраза BLANK WHEN ZERO используется

1) для числового редактируемого данного, оно будет содержать все пробелы, если значение передаваемого данного равно нулю;

2) для числового данного, его категория автоматически изменяется на числовую редактируемую независимо от строки-литер фразы PICTURE.

В качестве примеров двух описанных фраз можно привести следующие описания:

DATA DIVISION.

FILE SECTION

:

01 RECORD-X PICTURE IS X(25) JUSTIFIED RIGHT.

01 RECORD-Y.

05 ITEM-A PICTURE IS BB99.99 BLANK WHEN ZERO.

WORKING-STORAGE SECTION.

77 SINGLE-ITEM-X PICTURE IS 999.999.99 VALUE IS
3000.50 BLANK WHEN ZERO.

01 SINGLE-ITEM-Y PICTURE IS A(10) JUSTIFIED RIGHT.

Фраза BLANK WHEN ZERO не может появиться вместе с фразой PICTURE, в строке-литер которой присутствуют символы подавления нулей — звездочки.

Фраза SIGN используется только для тех числовых данных, которые имеют знак. Знак указывает, является ли значение данного положительным или отрицательным. В строке-литер фразы PICTURE данного со знаком должен присутствовать символ знака S, но наличие этого символа еще не определяет форму представления знака и его расположение в данном. Ранее в этой книге предлагалось совмещать знак с самой правой цифрой в один символ, так что данное со значением плюс 32 хранилось бы, например, в виде 3В

(или в еще каком-то аналогичном виде). Фраза SIGN предоставляет программисту метод определения положения и представления знака числа. Ее формальное определение таково:

$$\text{SIGN IS } \left\{ \begin{array}{c} \text{LEADING} \\ \text{TRAILING} \end{array} \right\} \text{SEPARATE CHARACTER}$$

Фразу SIGN можно использовать как для элементарного, так и для группового данного. В последнем случае она применяется к каждому из подчиненных данных, у которых в строке-литер фразы PICTURE присутствует символ S. К любому отдельному данному разрешается применять лишь одну фразу SIGN, так что вложение таких фраз не допускается. Эту фразу можно использовать двояко: вместе со словами SEPARATE CHARACTER (ОТДЕЛЬНО) и без них. Например:

05 NUMBER-ONE PICTURE IS S9(5) SIGN IS TRAILING.
05 NUMBER-TWO PICTURE IS S9(5) SIGN IS LEADING
SEPARATE CHARACTER.

В первом случае (SIGN IS LEADING (ЗНАК ПЕРВЫЙ) или SIGN IS TRAILING (ЗНАК ПОСЛЕДНИЙ)) знак совмещается либо с первой, либо с последней цифрой данного. В этом случае знак не занимает отдельной позиции литеры. Если бы значение данного NUMBER-ONE в приведенном выше примере было равно плюс 32, то данное состояло бы из пяти стандартных литер, например 0003B. Во втором случае (при использовании слов SEPARATE CHARACTER) знак хранится в отдельной позиции литеры и является либо первой (LEADING), либо последней (TRAILING) литерой. Для обозначения знака в этом случае используются литеры + или —. Если бы значение данного NUMBER-TWO было равно +32, то оно заняло бы шесть позиций: +00032. Данное, в описании которого встречается фраза SIGN, остается числовым и может быть использовано во всех арифметических вычислениях или сравнениях при определении истинности условия отношения; при этом о знаке числа можно не беспокоиться. Во время выполнения программы будут осуществляться все необходимые внутренние преобразования, отвечающие требованиям конкретной исходной машины (OBJECT-COMPUTER).

Упражнения

1. Добавьте описательные статьи-комментарии и строки комментариев в одну из программ предыдущих упражнений.
2. Напишите статью-описания-данного для группового данного, для подчиненных данных которого необходимо задать определенные начальные значения:

а) четыре подчиненных данных по пять цифр в каждом с отдельным знаком и одинаковыми начальными значениями, равными —41.

б) четыре подчиненных данных из пяти буквенно-цифровых цифр с одинаковыми начальными значениями из всех пробелов.

3. Напишите полную КОБОЛ-программу для чтения одной записи формата

```
01 INPUT-RECORD.
   05 INPUT-NAME      PICTURE IS A(10).
   05 VALUE-A          PICTURE IS S9(5).
   05 VALUE-B          PICTURE IS S9(5).
   05 VALUE-C          PICTURE IS S9(5).
   05 VALUE-D          PICTURE IS S9(5).
```

и сложения всех числовых данных с помещением суммы в выходную запись. Если сумма окажется отрицательной, в качестве ответа запишите все пробелы. Отдельно поместите в выходную запись имя (INPUT-NAME), сдвинутое вправо.

7.3. Уточнение имен

Имена, используемые в КОБОЛ-программе, должны быть однозначными, так чтобы обращение к ним в разделе процедур не приводило к двусмысленности. Однако возможно, а иногда и желательно, использовать одинаковое имя для различных данных. Для сохранения однозначности имен в этом случае служит *уточнение* имен, с помощью которого внешне одинаковые имена могут быть различены путем указания группы, к которой они принадлежат. Таким образом, имена-параграфов в пределах одной секции должны быть все различны, но они могут совпадать с именами-параграфов в других секциях, так как их однозначность может быть установлена с помощью уточнения имени-параграфа путем добавления имени-секции по правилу:

$$\text{имя-параграфа} \left\{ \begin{array}{c} \text{OF} \\ \text{IN} \end{array} \right\} \text{имя-секции}$$

В приведенном ниже примере используются две секции с именами FIRST и SECOND. Параграфы в этих секциях имеют одно и то же имя PARA-A. Эти параграфы можно различить с помощью уточнений

PARA-A IN FIRST

и

PARA-A IN SECOND

Уточнение использует имя-секции без самого слова SECTION (СЕКЦИЯ). Необходимо также отметить, что при обращении к параграфу, расположенному в той же секции, что и само обращение, уточнять имя-параграфа не требуется. Пример, о котором идет речь, таков:

PROCEDURE DIVISION.

FIRST SECTION.

PARA-A.

READ INPUT-FILE AT END GO TO PARA-A IN SECOND.

GO TO PARA-A.

SECOND SECTION.

PARA-A.

STOP RUN.

Возможность использования одинаковых имен-параграфов в различных секциях позволяет писать различные секции одной программы нескольким программистам, не заботясь при этом о том, чтобы все имена были различны. Возможно также использовать секции из одной программы в другой программе, не беспокоясь о том, что некоторые имена могут совпадать.

Допускается также уточнение имен, относящихся к данным. Такое уточнение используется даже более часто, чем уточнение имен-параграфов. Так же как и при использовании одинаковых имен-параграфов, одинаковые имена данных допускаются только в том случае, когда они могут быть точно идентифицированы с помощью имени группы, к которой они принадлежат. Уточненное-имя-данного определяется следующим образом:

$$\left\{ \begin{array}{l} \text{имя-данного} \\ \text{уточненное-имя-данного} \end{array} \right\} \left\{ \begin{array}{l} \text{OF} \\ \text{IN} \end{array} \right\} \text{имя-данного}$$

Так как уточненное-имя-данного, например ITEM-X OF FIELD-B, также может быть уточнено, может существовать целый ряд вложенных уточнений, например:

ITEM-X OF FIELD-B OF GROUP-L IN INPUT-RECORD.

Между уточняющими союзами OF (ИЗ) и IN (В) нет никакого различия. Они взаимозаменяемы. Требуются только те уточнения, которые достаточны для обеспечения однозначности имени, так что уточненные-имена-данных

ITEM-X OF FIELD-B

ITEM-X OF INPUT-RECORD

ITEM-X OF GROUP-L OF INPUT-RECORD

могут однозначно определять или нет нужное данное ITEM-X в зависимости от наличия других групповых данных с именами FIELD-B, GROUP-L или INPUT-RECORD. Например:

01 INPUT-RECORD.

05 GROUP-L.

10 FIELD-B.

15 ITEM-X

PICTURE IS X.

15 ITEM-Y

PICTURE IS X.

10 FIELD-C

PICTURE IS X(8).

05 GROUP-M.

10 ITEM-X

PICTURE IS 9(5).

10 FIELD-B

PICTURE IS X(2).

05 GROUP-N.

10 ITEM-X

PICTURE IS 9(5).

10 FIELD-C

PICTURE IS ++.9.

В этом примере строки-литер фраз PICTURE данных с одинаковыми именами сознательно выбраны различными с тем, чтобы подчеркнуть, что эти данные разные и что обсуждение касается только имен. Имя данного ITEM-Y единственно и не нуждается в уточнении, хотя и в этом случае уточнение не является ошибкой. Оно просто не является необходимым. Имена ITEM-X, FIELD-B и FIELD-C при наличии данного выше описания всегда должны уточняться, например:

FIELD-B IN GROUP-L

FIELD-B IN GROUP-M

FIELD-C IN GROUP-L

FIELD-C IN GROUP-N

ITEM-X IN FIELD-B OF GROUP-L

ITEM-X IN GROUP-L

ITEM-X IN GROUP-M

ITEM-X IN GROUP-N

Если другие записи содержат такие же имена, то необходимо уточнение с помощью имени записи INPUT-RECORD. Обратите внимание, что в рассматриваемом примере групповые данные уровня 05 не могли бы иметь одинаковых имен, так как не было бы возможности различить их. Более того, одинаковые имена не могут появляться на различных уровнях одной и той же иерархии имен так, что при уточнении имя уточняло бы само себя. Уточнение вида

ITEM-Q IN ITEM-Q

(НЕВЕРНО!)

недопустимо.

Имена данных с номерами-уровней 01 или 77 никогда не могут быть уточнены, так как они не могут принадлежать какой-либо другой группе. Уточненное-имя-данного будет в дальнейшем называться также идентификатором.

Один важный случай использования уточненных-имен-данных описывается в следующем разделе.

7.4. Вариант CORRESPONDING

Наличие уточнения имен позволяет программистам использовать одинаковые имена в различных записях, сохраняя возможность однозначной идентификации каждого данного. Однако наиболее важное применение уточнение имен находит при использовании варианта CORRESPONDING (СООТВЕТСТВЕННО). Этот вариант допустим для оператора MOVE (ПОМЕСТИТЬ) и для арифметических операторов ADD (СЛОЖИТЬ) и SUBTRACT (ОТНЯТЬ). Этот вариант операторов всегда включает два групповых данных, причем соответствующая операция применяется к парам данных, имеющих одинаковые имена и расположенных в разных группах. Под *одинаковыми именами* понимаются имена, полученные с учетом уточнения внутри групп (без уточнения с помощью имени самой группы). Данные в таких парах называются *соответствующими данными*. Например, в следующих двух группах (с опущенными фразами PICTURE):

01 GROUP-A.

05 ITEM-A.

05 ITEM-B.

05 ITEM-C.

01 GROUP-B.

05 ITEM-A.

05 ITEM-B.

05 ITEM-C.

данные с именами ITEM-A, ITEM-B и ITEM-C образуют пары соответствующих данных. Вместо того, чтобы писать операторы

MOVE ITEM-A OF GROUP-A TO ITEM-A OF GROUP-B.

MOVE ITEM-B OF GROUP-A TO ITEM-B OF GROUP-B.

MOVE ITEM-C OF GROUP-A TO ITEM-C OF GROUP-B.

можно достигнуть того же результата с помощью одного оператора

MOVE CORRESPONDING GROUP-A TO GROUP-B.

Согласование соответствующих данных не обязательно должно быть таким же простым, как в приведенном примере. Эти данные могут быть расположены в любом порядке, и, кроме того, могут существовать данные, не входящие в пары, т. е. не имеющие соответствующих данных в другой записи. Вариант CORRESPONDING применяется только к данным с одинаковыми именами, ос-

тальные данные игнорируются. Используя пример записи INPUT-RECORD из предыдущего раздела, получим, что операторы

MOVE CORRESPONDING GROUP-M TO GROUP-N

и

MOVE ITEM-X OF GROUP-M TO ITEM-X OF GROUP-N

приводят к одному и тому же результату, так как данные ITEM-X OF GROUP-M и ITEM-X OF GROUP-N образуют единственную пару соответствующих данных. Заметьте, что одинаковые имена не включают имена групповых данных, используемые в варианте CORRESPONDING.

Формальное определение этого варианта оператора MOVE следующее:

$$\text{MOVE} \left\{ \begin{array}{c} \text{CORRESPONDING} \\ \text{CORR} \end{array} \right\} \text{идентификатор-1 TO идентификатор-2}$$

где слово CORR (COOTB) является сокращением слова CORRESPONDING (COOTBETCTBENHO), а идентификатор-1 и идентификатор-2 должны ссылаться на групповые данные. При использовании варианта CORRESPONDING справа от слова TO (B) может быть указано лишь одно имя-данного. Это отличает рассматриваемый вариант от описанного ранее варианта оператора MOVE допущавшего, например, операторы вида

MOVE "N" TO ABLE-X BAKER-X CHARLIE-X.

При использовании варианта CORRESPONDING оператора MOVE по крайней мере одно данное из пары соответствующих данных должно быть элементарным. Редактирование, выполняемое при передаче данных, осуществляется в соответствии с фразами PICTURE принимающих данных. Напомним, что если одно из данных в операторе MOVE является групповым, то передача данных осуществляется так, как если бы оба данных были буквенно-цифровыми.

Существуют также варианты CORRESPONDING операторов ADD и SUBTRACT, имеющие следующие форматы:

1. сложение:

$$\text{ADD} \left\{ \begin{array}{l} \text{CORRESPONDING} \\ \text{CORR} \end{array} \right\} \text{идентификатор-1 TO идентификатор-2}$$

[ROUNDED]
[ON SIZE ERROR повелительные-операторы]

2. вычитание:

$$\text{SUBTRACT} \left\{ \begin{array}{l} \text{CORRESPONDING} \\ \text{CORR} \end{array} \right\} \text{идентификатор-1 FROM идентификатор-2}$$

[ROUNDED]
[ON SIZE ERROR повелительные-операторы]

Для операторов умножения, деления и для остальных операторов КОБОЛа варианта CORRESPONDING не существует. Все данные, относящиеся к парам соответствующих данных в арифметических операторах, должны быть элементарными числовыми данными. Вариант GIVING (ПОЛУЧАЯ) не допускается. Значение каждого данного из группового данного, заданного идентификатором-1, добавляется к соответствующему данному из группового данного, заданного идентификатором-2, или вычитается из него. При этом результат помещается на место второго данного. Если хотя бы один из результатов операций над парами соответствующих данных превзойдет максимальное значение, допустимое фразой PICTURE, относящейся к данному, в которое записывается результат, будут выполняться повелительные-операторы, предусмотренные во фразе ON SIZE ERROR (ПРИ ПЕРЕПОЛНЕНИИ). В качестве соответствующих данных выбираются только те данные из двух групп, которые на всех уровнях уточнения имеют одинаковые описания. Например, рассмотрим следующие две статьи-описания-записи:

01 MAJOR-A.

05 MINOR-G.

10 ITEM-X

PICTURE IS X.

10 SMALL-A

PICTURE IS X.

05 MINOR-H.

10 SMALL-B

PICTURE IS X.

10 SMALL-A

PICTURE IS X.

05 MINOR-I.

10 ITEM-X

PICTURE IS X.

10 SMALL-A

PICTURE IS X.

01 MAJOR-B.

05 MINOR-F.

10 ITEM-X	PICTURE IS X.
10 SMALL-A	PICTURE IS X.
05 MINOR-G.	
10 SMALL-A	PICTURE IS X.
10 SMALL-B	PICTURE IS X.
05 MINOR-H.	
10 SMALL-A	PICTURE IS X.
10 SMALL-B	PICTURE IS X.
05 MINOR-I.	
10 ITEM-X	PICTURE IS X.
10 ITEM-Y	PICTURE IS X.

В обеих группах имеется четырнадцать элементарных данных, но из них составляются только четыре пары соответствующих данных. Следовательно оператор

MOVE CORRESPONDING MAJOR-A TO MAJOR-B

будет выполнять те же действия, что и группа операторов

```

MOVE SMALL-A IN MINOR-G OF MAJOR-A
  TO SMALL-A IN MINOR-G OF MAJOR-B.
MOVE SMALL-A IN MINOR-H OF MAJOR-A
  TO SMALL-A IN MINOR-H OF MAJOR-B.
MOVE SMALL-B IN MINOR-H OF MAJOR-A
  TO SMALL-B IN MINOR-H OF MAJOR-B.
MOVE ITEM-X IN MINOR-I OF MAJOR-A
  TO ITEM-X IN MINOR-I OF MAJOR-B.

```

Все данные являются элементарными, так что оператор MOVE допустим. Ни одно из элементарных данных не является числовым, поэтому рассматриваемые групповые данные не могут использоваться в вариантах CORRESPONDING арифметических операторов.

7.5. Пример программы

IDENTIFICATION DIVISION.

PROGRAM-ID. PROG750.

AUTHOR.

ИМЯ ПРОГРАММИСТА.

INSTALLATION.

НАЗВАНИЕ И МЕСТОНАХОЖДЕНИЕ

ОПИСАТЕЛЬНОГО МАТЕРИАЛА.

DATE-WRITTEN.

ПЕРВАЯ ДАТА НАПИСАНИЯ.

ДАТЫ ВНЕСЕНИЯ ИЗМЕНЕНИЙ.

DATE-COMPILED.

ДАННАЯ СТАТЬЯ БУДЕТ ЗАМЕНЕНА ТЕКУЩЕЙ
ДАТОЙ КОМПИЛЯЦИИ.

SECURITY.

ПРИМЕР СТАТЬИ-КОММЕНТАРИЯ ФРАЗЫ SECURITY
(ПОЛНОМОЧИЯ):

ВСЕ ОСНОВНЫЕ ФАЙЛЫ (MASTER FILES)

ДОЛЖНЫ ХРАНИТЬСЯ В СЕЙФЕ БИБЛИОТЕКИ
ЛЕНТ И УДАЛЯТЬСЯ ТОЛЬКО ПО РАСПИСКЕ.

ЭТОТ ПРИМЕР ПРОГРАММЫ БЫЛ НАПИСАН ДЛЯ
ИСПОЛЬЗОВАНИЯ БОЛЬШИНСТВА НОВЫХ
ВОЗМОЖНОСТЕЙ, ОПИСАННЫХ В ГЛАВАХ ШЕСТЬ
И СЕМЬ.

ЗАДАЧА СОСТОИТ В СЛЕДУЮЩЕМ:

ЗАДАНА ЗАПИСЬ ВХОДНОГО ФАЙЛА (INPUT-FILE)
С ИМЕНЕМ SELECT RECORD, В КОТОРОЙ УКАЗАН
НОМЕР (ITEM-NUMBER) ОПРЕДЕЛЕННОГО ИЗДЕЛИЯ.
НЕОБХОДИМО НАЙТИ В ДВУХ ОТДЕЛЬНЫХ
ОСНОВНЫХ ФАЙЛАХ ЗАПИСИ, ИМЕЮЩИЕ
УКАЗАННЫЙ НОМЕР, И СКОПИРОВАТЬ
ОПРЕДЕЛЕННУЮ ИНФОРМАЦИЮ ИЗ ЭТИХ ЗАПИСЕЙ
В ДРУГОЙ ФАЙЛ (EXTRACT-FILE).

ОБА ОСНОВНЫХ ФАЙЛА ХРАНЯТСЯ НА ОДНОЙ
КАТУШКЕ МАГНИТНОЙ ЛЕНТЫ. НЕ ПЕРЕМАТЫВАЯ
(WITH NO REWIND) ФАЙЛ С ВЫБРАННЫМИ
ЗАПИСЯМИ (EXTRACT-FILE), А ЧИТАЯ ЕГО
РЕВЕРСНО (REVERSED), РАСПЕЧАТАЙТЕ ВСЕ
ЗАПИСИ, В КОТОРЫХ ОБЪЕМ ПРОДАЖИ (SALES-
VOLUME) МЕНЬШЕ, ЧЕМ СРЕДНИЙ ОБЪЕМ ПРОДАЖИ
(AVERAGE-SALES-VOLUME) ДЛЯ ЭТОГО ФАЙЛА.

БЛОК-СХЕМА ПРЕДЛАГАЕМОЙ ПРОЦЕДУРЫ
ПРИВЕДЕНА НА РИС. 7.1.

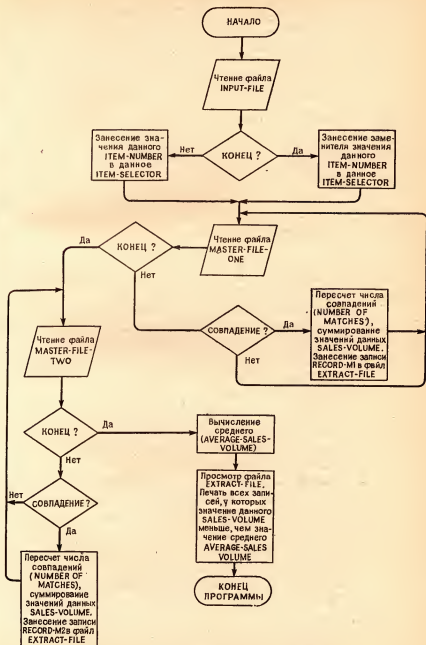


Рис. 7.1. Блок-схема примера программы.

ПАРАГРАФ SPECIAL-NAMES ОПРЕДЕЛЯЕТ ФУНКЦИЮ (C01), КОТОРАЯ ПРОПУСКАЕТ МЕСТО ДО НАЧАЛА СЛЕДУЮЩЕЙ СТРАНИЦЫ. СЛОВО "C01" ОПРЕДЕЛЕНО КОНКРЕТНОЙ РЕАЛИЗАЦИЕЙ.

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER. COMPUTER-NAME.

OBJECT-COMPUTER. COMPUTER-NAME.

SPECIAL-NAMES.

 C01 IS TO-NEXT-PAGE.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

 SELECT INPUT-FILE

 ASSIGN TO READER

 RESERVE 1 AREA

 ACCESS MODE IS SEQUENTIAL

 ORGANIZATION IS SEQUENTIAL.

 SELECT MASTER-FILE-ONE

 ASSIGN TO TAPE

 RESERVE 2 AREAS

 ACCESS MODE IS SEQUENTIAL

 ORGANIZATION IS SEQUENTIAL.

 SELECT MASTER-FILE-TWO

 ASSIGN TO TAPE

 RESERVE 2 AREAS

 ACCESS MODE IS SEQUENTIAL

 ORGANIZATION IS SEQUENTIAL.

 SELECT EXTRACT-FILE

 ASSIGN TO TAPE

 RESERVE 2 AREAS

 ACCESS MODE IS SEQUENTIAL

 ORGANIZATION IS SEQUENTIAL.

 SELECT PRINT-FILE

 ASSIGN TO PRINTER

 RESERVE 2 AREAS

ACCESS MODE IS SEQUENTIAL
ORGANIZATION IS SEQUENTIAL.

I-O-CONTROL.

SAME AREA FOR MASTER-FILE-ONE

MASTER-FILE-TWO INPUT-FILE.

SAME RECORD AREA FOR PRINT-FILE

EXTRACT-FILE.

MULTIPLE FILE TAPE CONTAINS

MASTER-FILE-ONE POSITION 1

MASTER-FILE-TWO POSITION 2.

DATA DIVISION.

FILE SECTION.

FD INPUT-FILE

LABEL RECORDS ARE STANDARD

RECORD CONTAINS 80 CHARACTERS

BLOCK CONTAINS 80 CHARACTERS

DATA RECORD IS SELECT-RECORD.

01 SELECT-RECORD.

05 ITEM-NUMBER PICTURE IS X(5).

05 FILLER PICTURE IS X(75).

FD MASTER-FILE-ONE.

LABEL RECORDS ARE STANDARD

RECORD CONTAINS 40 CHARACTERS

BLOCK CONTAINS 4000 CHARACTERS

DATA RECORD IS RECORD-M1.

01 RECORD-M1.

05 ITEM-NUMBER PICTURE IS X(5).

05 SALES-TERRITORY PICTURE IS X(25).

05 SALES-VOLUME PICTURE IS 9(6).

05 DOLLAR-SALES PICTURE IS 9(4).

FD MASTER-FILE-TWO

LABEL RECORDS ARE STANDARD

RECORD CONTAINS 44 CHARACTERS

BLOCK CONTAINS 4400 CHARACTERS
DATA RECORD IS RECORD-M2.

01 RECORD-M2.

05 RECORD-CODE	PICTURE IS X.
05 SALES-TERRITORY	PICTURE IS X(25).
05 SALES-VOLUME	PICTURE IS 9(6).
05 SALES-PERSON-CODE	PICTURE IS X(3).
05 DOLLAR-SALES	PICTURE IS 9(4).
05 ITEM-NUMBER	PICTURE IS X(5).

FD EXTRACT-FILE

LABEL RECORDS ARE STANDARD
RECORD CONTAINS 40 CHARACTERS
BLOCK CONTAINS 4000 CHARACTERS
DATA RECORD IS EXTRACT-RECORD.

01 EXTRACT-RECORD.

05 FILLER	PICTURE IS X.
05 SALES-TERRITORY	PICTURE IS X(25) JUSTIFIED RIGHT.
05 SALES-VOLUME	PICTURE IS 9(6).
05 DOLLAR-SALES	PICTURE IS \$,,\$\$.00 BLANK WHEN ZERO.

FD PRINT-FILE

LABEL RECORDS ARE STANDARD
RECORD CONTAINS 40 CHARACTERS
BLOCK CONTAINS 4000 CHARACTERS
DATA RECORD IS PRINT-LINE.

01 PRINT-LINE.

05 FILLER	PICTURE IS X.
05 SALES-TERRITORY	PICTURE IS X(25).
05 SALES-VOLUME	PICTURE IS 9(6).
05 DOLLAR-SALES	PICTURE IS \$,,\$\$.00.

WORKING-STORAGE SECTION.

77 ITEM-SELECTOR	PICTURE IS X(5).
77 NUMBER-OF-MATCHES	PICTURE IS 9(15) VALUE IS ZERO.
77 SUM-OF-VOLUMES	PICTURE IS 9(15) VALUE IS ZERO.

01 HEADER-MESSAGE.

05 FILLER PICTURE IS X VALUE IS SPACE.
05 FILLER PICTURE IS X(25)
 VALUE IS "AVERAGE SALES VOLUME IS".
05 AVERAGE-SALES-VOLUME PICTURE IS 9(6).
05 FILLER PICTURE IS X(8)
 VALUE IS ALL SPACES.

PROCEDURE DIVISION.

READ-INPUT-FILE SECTION.

START-PARAGRAPH.

OPEN INPUT INPUT-FILE.
READ INPUT-FILE RECORD AT END MOVE "99350"
TO ITEM-SELECTOR
GO TO START-PARAGRAPH OF
CREATE-EXTRACT-FILE.

- * ЕСЛИ ЗАПИСЬ ВХОДНОГО ФАЙЛА (INPUT-FILE
- * RECORD) НЕ ИСПОЛЬЗУЕТСЯ, ТО ПРОГРАММА
- * ВЫБЕРЕТ В КАЧЕСТВЕ ЗАМЕНИТЕЛЯ ЗНАЧЕНИЯ
- * ДАННОГО ITEM-NUMBER ЗНАЧЕНИЕ 99350.
- MOVE ITEM-NUMBER IN SELECT-RECORD TO ITEM-
- SELECTOR.
- * ОБРАТИТЕ ВНИМАНИЕ, ЧТО ПОСЛЕ ОКОНЧАНИЯ
- * ДАННОЙ СЕКЦИИ СРАЗУ ЖЕ ИДЕТ ИМЯ НОВОЙ
- * СЕКЦИИ; ОПЕРАТОР "GO TO" ДЛЯ СВЯЗЫВАНИЯ
- * СМЕЖНЫХ СЕКЦИЙ НЕ НУЖЕН, ТАК КАК ПЕРЕХОД
- * ИЗ ПЕРВОЙ СЕКЦИИ ВО ВТОРУЮ ОСУЩЕСТВЛЯЕТСЯ
- * С ПОМОЩЬЮ НОРМАЛЬНОГО ПОТОКА
- * УПРАВЛЕНИЯ.

CREATE-EXTRACT-FILE SECTION.

START-PARAGRAPH.

CLOSE INPUT-FILE.
OPEN INPUT MASTER-FILE-ONE.
OPEN OUTPUT EXTRACT-FILE.
MOVE ZERO TO NUMBER-OF-MATCHES
SUM-OF-VOLUMES.

* ЗАМЕЬТЕ, ЧТО ПРЕДЫДУЩИЙ ОПЕРАТОР MOVE
* ЯВЛЯЕТСЯ ЛИШНИМ, ТАК КАК НАЧАЛЬНЫЕ
* ЗНАЧЕНИЯ СООТВЕТСТВУЮЩИХ ДАННЫХ БЫЛИ
* ЗАДАНЫ С ПОМОЩЬЮ ФРАЗ VALUE IS.

READ-ONE-LOOP.

READ MASTER-FILE-ONE RECORD AT END GO TO
READ-SECOND FILE.

IF ITEM-NUMBER IN RECORD-M1 IS EQUAL TO
ITEM-SELECTOR

ADD 1 TO NUMBER-OF-MATCHES

ADD SALES-VOLUME OF RECORD-M1 TO
SUM-OF-VOLUMES

MOVE CORRESPONDING RECORD-M1 TO
EXTRACT-RECORD

WRITE EXTRACT-RECORD.

GO TO READ-ONE-LOOP.

* ПЕРВЫЙ ЦИКЛ ЧТЕНИЯ (READ-ONE-LOOP)
* ВЫДЕЛЯЕТ ИЗ ПЕРВОГО ОСНОВНОГО ФАЙЛА
* (MASTER-FILE-ONE) ВСЕ ЗАПИСИ, У КОТОРЫХ
* ЗНАЧЕНИЕ ДАННОГО ITEM-NUMBER
* СОВПАДАЕТ СО ЗНАЧЕНИЕМ ДАННОГО
* ITEM-SELECTOR.

READ-SECOND-FILE.

CLOSE MASTER-FILE-ONE WITH LOCK.

OPEN INPUT-MASTER-FILE-TWO.

READ-TWO-LOOP.

READ MASTER-FILE-TWO RECORD AT END GO TO
PRINT-LOW-VOLUME.

IF ITEM-NUMBER IN RECORD-M2 IS EQUAL TO
ITEM-SELECTOR

ADD 1 TO NUMBER-OF-MATCHES

ADD SALES-VOLUME OF RECORD-M2 TO
SUM-OF-VOLUMES

MOVE CORRESPONDING RECORD-M2 TO
EXTRACT-RECORD

WRITE EXTRACT-RECORD.

GO TO READ-TWO-LOOP.

* ФОРМАТ ЗАПИСИ RECORD-M2 ОТЛИЧАЕТСЯ ОТ
* ФОРМАТА ЗАПИСИ RECORD-M1, НО
* ПРИ ВЫПОЛНЕНИИ ОПЕРАТОРА "MOVE
* CORRESPONDING" В ЗАПИСЬ EXTRACT-RECORD
* БУДУТ ПЕРЕДАНЫ ВСЕ НЕОБХОДИМЫЕ ДАННЫЕ
* КАК В ТОМ, ТАК И В ДРУГОМ СЛУЧАЕ.

PRINT-LOW-VOLUME SECTION.

REMARKS-PARAGRAPH.

* ТЕПЕРЬ ФАЙЛ EXTRACT-FILE СОДЕРЖИТ ЗАПИСИ
* ИЗ ОБОИХ ОСНОВНЫХ ФАЙЛОВ (MASTER-FILE-ONE
* И MASTER-FILE-TWO), ИМЕЮЩИЕ ОДИНАКОВОЕ
* ЗНАЧЕНИЕ ДАННОГО ITEM-NUMBER. ПОСЛЕ
* ПОДСЧЕТА СРЕДНЕГО ОБЪЕМА ПРОДАЖИ
* (AVERAGE-SALES-VOLUME) ЭТОТ ФАЙЛ БУДЕТ
* ПРОЧИТАН И ТЕ ИЗ ЗАПИСЕЙ, У КОТОРЫХ ОБЪЕМ
* ПРОДАЖИ (SALES-VOLUME) МЕНЬШЕ, ЧЕМ СРЕДНИЙ
* ОБЪЕМ ПРОДАЖИ, БУДУТ РАСПЕЧАТАНЫ.
* ДЛЯ ЭКОНОМИИ ВРЕМЕНИ ПЕРЕМОТКИ ФАЙЛ
* EXTRACT-FILE БУДЕТ ЧИТАТЬСЯ РЕВЕРСНО
* (REVERSED). ЭТОТ ФАЙЛ БЫЛ ОТКРЫТ КАК
* ВЫХОДНОЙ (OUTPUT); ЕГО НЕОБХОДИМО
* ЗАКРЫТЬ, А ЗАТЕМ ОТКРЫТЬ ВНОВЬ КАК
* ВХОДНОЙ (INPUT).

START-PARAGRAPH.

DIVIDE NUMBER-OF-MATCHES INTO SUM-OF-
VOLUMES GIVING AVERAGE-SALES-VOLUME.
CLOSE MASTER-FILE-TWO WITH LOCK
EXTRACT-FILE WITH NO REWIND.
OPEN INPUT EXTRACT-FILE REVERSED OUTPUT
PRINT-FILE.

WRITE PRINT-LINE FROM HEADER-MESSAGE
BEFORE ADVANCING TO-NEXT-PAGE.

* ОБРАТИТЕ ВНИМАНИЕ, ЧТО ЗНАЧЕНИЕ СЛОВ
* TO-NEXT-PAGE БЫЛО ОПРЕДЕЛЕНО В ПАРАГРАФЕ
* SPECIAL-NAMES.

READ-LOOP.

READ EXTRACT-FILE RECORD AT END CLOSE
EXTRACT-FILE
PRINT-FILE STOP RUN.

IF SALES-VOLUME IN EXTRACT-RECORD IS LESS
THAN AVERAGE-SALES-VOLUME
WRITE PRINT-LINE BEFORE ADVANCING
2 LINES.

GO TO READ-LOOP.

* ПРИ РЕВЕРСНОМ ЧТЕНИИ ОПЕРАТОР READ
* НЕ ИЗМЕНЯЕТСЯ, НАПРАВЛЕНИЕ ЧТЕНИЯ.
* БЫЛО ОПРЕДЕЛЕНО В ОПЕРАТОРЕ OPEN;
* ЗАПИСЬ ЧИТАЕТСЯ В ОБЛАСТЬ ЗАПИСИ
* НОРМАЛЬНЫМ ОБРАЗОМ, НЕСМОТРИ НА ТО
* ЧТО ЛЕНТА ДВИЖЕТСЯ В ОБРАТНОМ
* НАПРАВЛЕНИИ. СТРОКА КОММЕНТАРИЯ
* НЕ МОЖЕТ БЫТЬ ПОСЛЕДНЕЙ СТРОКОЙ
* ПРОГРАММЫ, ПОЭТОМУ ДОБАВЛЕН
* СЛЕДУЮЩИЙ НЕИСПОЛЬЗУЕМЫЙ ПАРАГРАФ.

END PARAGRAPH.

STOP RUN.

Упражнения

1. Напишите полную КОБОЛ-программу для печати отчета с анализом продаж, получаемых из трех отдельных файлов с последовательным доступом, каждый из которых упорядочен по номерам покупателей (CUSTOMER-NUMBER). Единственный смысл такого упорядочения заключается в том, что записи с одним и тем же номером покупателя в каждом файле расположены подряд. Подсчитайте объемы продаж в данной местности (LOCAL SALES), в области (REGIONAL SALES) и в общегосударственном масштабе (NATIONAL SALES) для каждого номера покупателя и распечатайте эту информацию в виде:

CUSTOMER-NUMBER	LOCAL SALES	REGIONAL SALES
9(5)	ZZ,ZZZ.99	ZZ,ZZZ.99
	NATIONAL SALES	
	ZZZ,ZZZ.99	

Подсчитайте общие суммы этих объемов для всех различных (и приемлемых) номеров покупателей и напечатайте эти итоговые суммы в конце отчета в виде

FINAL TOTALS \$ZZZ,ZZ.99 \$ZZZ,ZZZ.99 \$Z,ZZZ,ZZZ.99

Приемлемыми номерами покупателей считаются те, которые удовлетворяют следующим двум требованиям:

а) они относятся к информации, имеющей возраст не больше года;

б) они являются полностью числовыми.

Подсчитайте общие суммы для всех неприемлемых записей и распечатайте их в качестве самой последней строки отчета в виде:

OTHER TOTALS \$ZZZ,ZZZ.99 \$ZZZ,ZZZ.99 \$Z,ZZZ,ZZZ.99

Для рассматриваемой задачи составьте возможную статью-описания-записи для входных записей, а также для каждой из выходных строк. Позаботьтесь о том, чтобы заголовок страницы печатался на каждой новой странице отчета.

7.6. Различные описания одного и того же данного

Существуют еще две фразы, которые могут быть использованы в статьях-описания-данного: фраза RENAMES (ПЕРЕИМЕНОВЫВАЕТ) и фраза REDEFINES (ПЕРЕОПРЕДЕЛЯЕТ). Основное назначение этих двух фраз заключается в том, чтобы позволить различным образом определять одну и ту же область внутренней памяти. Ни одна из этих фраз не может быть использована для переопределения описания статьи уровня 01 в секции файлов. Для каждой данной записи может указываться по нескольку тех и других фраз в различных комбинациях, что позволяет задавать большое разнообразие различных определений для одной и той же области памяти. Первой будет рассмотрена фраза RENAMES, но порядок, в котором располагаются эти фразы, не имеет никакого значения.

Фраза RENAMES

Фраза RENAMES может использоваться только в специальной статье-описания-данного, имеющей номер-уровня 66. При использовании таких статей они должны следовать непосредственно за последней статьей-описания в записи. Статья уровня 66 связывает новые имена с ранее определенными статьями, она не имеет никакого отношения к значениям или типу статей. Ее формальное определение таково:

66 переименовывающее-имя-данного RENAMES.имя-данного-1 [THRU имя-данного-2]

Номер уровня должен располагаться в поле А, а остальная часть статьи — в поле В. Статьи уровня 66 должны заканчиваться точкой, за которой следует пробел. Обратите внимание, что используются имена-данных, а не идентификаторы. Имя-данного-1 и имя-данного-2 должны быть именами данных предшествующей записи и не могут указываться с индексами или именами-индексов, например:

```
01 MAJOR-A
    05 SMALL-A    PICTURE IS X.
    05 ITEM-Y     PICTURE IS X.
    05 ITEM-Z     PICTURE IS X.
66 ITEM-X RENAMES SMALL-A.
```

Для задания нескольких новых имен одной и той же исходной статьи могут использоваться дополнительные фразы RENAMES:

```
66 ITEM-X RENAMES SMALL-A.
66 NEW-X RENAMES SMALL-A.
66 NOVEL-Z RENAMES SMALL-A.
```

Однако само переименовывающее-имя-данного не может быть переименовано, так что следующее недопустимо:

```
66 ITEM-X RENAMES SMALL-A.
66 NEXT-A RENAMES ITEM-X.      (НЕВЕРНО!)
```

Вообще, статья уровня 66 не может переименовывать статьи уровней 66, 77 и 01. Если данное с именем имя-данного-1 является элементарным, а имя-данного-2 не используется, то переименовывающее-имя-данного относится к элементарному данному. Если данное с именем имя-данного-1 является групповым или используется имя-данного-2, то переименовывающее-имя-данного относится к групповому данному. Использование имени-данного-2 дает возможность группировать несколько данных под новым именем. Существует несколько ограничений на использование имени-данного-2:

- 1) имя-данного-2 не может совпадать с именем-данного-1;
- 2) имя-данного-2 не может быть подчиненным имени-данного-1;
- 3) в статье-описания-данного имя-данного-2 должно быть расположено после имени-данного-1.

Пусть задана статья-описания-данного

```
01 MAJOR-B.
    05 SMALL-A    PICTURE IS X.
    05 MINOR-G.
```

10 ITEM-X PICTURE IS X.

10 ITEM-Y PICTURE IS X.

Тогда следующие статьи уровня 66 являются недопустимыми:

66 NAME-A RENAMES SMALL-A THRU SMALL-A.
(НЕВЕРНО!)

66 NAME-B RENAMES ITEM-Y THRU SMALL-A.
(НЕВЕРНО!)

66 NAME-C RENAMES MINOR-G THRU ITEM-Y.
(НЕВЕРНО!)

Имя-данного-1 и имя-данного-2 могут относиться как к элементарным, так и к групповым данным. Данное с именем переименовывающее-имя-данного включает все данные

1) начиная с данного с именем имя-данного-1, если оно элементарное, или начиная с первого элементарного данного в данном с именем имя-данного-1, если оно групповое;

2) заканчивая данным с именем имя-данного-2, если оно элементарное, или заканчивая последним элементарным данным в данном с именем имя-данного-2, если оно групповое.

Например, если задана статья

66 NEW-A RENAMES SMALL-A THRU MINOR-G.

то данное с именем NEW-A будет групповым, включая элементарные данные SMALL-A, ITEM-X и ITEM-Y.

Имя-данного-1 и имя-данного-2 могут быть уточнены или сами могут быть использованы в качестве уточнителей, но переименовывающее-имя-данного не может быть уточнителем и может уточняться только с помощью имен статей уровня 01 или FD (ОФ). Статьи уровня 66 игнорируются вариантом CORRESPONDING, так что нельзя переименовать данное с тем, чтобы оно смогло образовать пару соответствующих данных с каким-либо данным другой записи. Таким образом, в описаниях

01 MAJOR-A.

05 SMALL-A PICTURE IS X.

05 ITEM-Y PICTURE IS X.

05 ITEM-Z PICTURE IS X.

66 ITEM-X RENAMES SMALL-A.

01 MAJOR-B.

05 ITEM-X PICTURE IS X.

05 ITEM-Y PICTURE IS X.

05 ITEM-Z PICTURE IS X.

фраза RENAMEs для записи MAJOR-A верна, но она не позволяет осуществить передачу данного SMALL-A в данное ITEM-X записи MAJOR-B с помощью оператора

MOVE CORRESPONDING MAJOR-A TO MAJOR-B.

Фраза REDEFINES

Фраза RENAMEs позволяет различным образом группировать смежные данные под новыми именами. Фраза REDEFINES (ПЕРЕОПРЕДЕЛЯЕТ) позволяет изменять структуру области записи, включая изменение типа фразы PICTURE. Существуют и другие отличия. Фраза RENAMEs появляется только в специальной статье уровня 66, которая должна быть написана сразу же после конца статьи-описания-данного. С другой стороны, фраза REDEFINES появляется непосредственно следом за данным, к которому она относится. Формальное определение фразы REDEFINES имеет вид:

номер-уровня переопределяющее-имя-данного REDEFINES
имя-данного фразы-переопределения

Номер-уровня должен иметь то же значение, что и номер-уровня имени-данного. Номер-уровня не может быть равен 01 в секции файлов и не может быть равен 66, т. е. ни запись файла, ни статья RENAMEs не могут быть переопределены. Обратите внимание, что статья-описания-данного уровня 77 может быть переопределена, но ее не разрешалось переименовывать. Могут быть переопределены статьи-описания-данного и уровня 01, но только в секции рабочей-памяти. Переопределение фактически описывается набором фраз-переопределения, следующих за именем-данного. Простой пример:

01 MAJOR-A.

05 MINOR-G PICTURE IS A (3).

05 SMALL-G REDEFINES MINOR-G PICTURE IS 9 (3).

Переопределение является дополнительным определением; оно не заменяет исходного определения. При этом никакой дополнительной памяти не выделяется. REDEFINES — это не глагол действия, и никакого изменения значения данного не происходит. Все определения и переопределения действительны одновременно. В примере с данным MAJOR-A элементарное данное MINOR-G является буквенным, а элементарное данное SMALL-G — числовым. Каждое имя-данного описывает одно и то же поле физической памяти, и в каждый момент времени это поле может быть занято только одним набором литер. Если существует другое числовое данное ITEM-X с фразой PICTURE IS 9(3), то оператор

MOVE ITEM-X TO MINOR-G (НЕВЕРНО!)

является недопустимым, так как числовое целое нельзя помещать в буквенное данное; а оператор

MOVE ITEM-X TO SMALL-G

правильен и вызовет передачу данного. Фраза REDEFINES не повлияет на данное, хранящееся в соответствующем поле; отдельные фразы PICTURE нескольких фраз REDEFINES будут влиять на передачу данного в это поле. Таким образом, при наличии описаний

01 RECORD-X.

05 SPACE-A PICTURE IS X(4).

05 SPACE-B REDEFINES SPACE-A PICTURE IS 99.9.

05 SPACE-C REDEFINES SPACE-A PICTURE IS \$(4).

01 VALUE-X VALUE IS 0078 PICTURE IS X(4).

помещение значения 0078 в данное RECORD-X привело бы к различным результатам в зависимости от используемого переопределения, например:

Оператор	Результат
MOVE VALUE-X TO SPACE-A.	0078
MOVE VALUE-X TO SPACE-B.	78.0
MOVE VALUE-X TO SPACE-C.	\$78

Обычно с помощью оператора MOVE нельзя передавать данное само в себя, так что оператор

MOVE ITEM-A TO ITEM-A (НЕВЕРНО!)

недопустим, но использование фразы REDEFINES позволяет выполнить такую передачу. Например, при наличии описаний:

05 DATA-ONE PICTURE IS S9(5).

05 DATA-TWO REDEFINES DATA-ONE PICTURE IS 9(5).

можно выполнить оператор

MOVE DATA-ONE TO DATA-TWO

который заменит данное со знаком на данное без знака. Однако часто перемещение данного в данное, переопределяющее его самого, может привести к непредсказуемым результатам, и этого следует, как правило, избегать. С фразой REDEFINES надо обходиться о-чень аккуратно еще и потому, что глагол REDEFINES не является

глаголом действия. Результаты работы следующего фрагмента программы оказались бы непредсказуемы:

05 ITEM-A PICTURE IS A(3).

05 ITEM-B REDEFINES ITEM-A PICTURE IS 9(3).

MOVE "ABC" TO ITEM-A.

ADD 32 TO ITEM-B.

(НЕВЕРНО!)

Фразы-переопределения могут представлять собой любые допустимые комбинации фраз описания-данных и могут содержать дополнительные номера-уровней. Ограничение заключается в том, что число литер, задаваемое в переопределении, должно в точности соответствовать полю, отведенному под данное в исходном определении. Использование специального данного FILLER (ЗАПОЛНИТЕЛЬ) могло бы помочь в устранении различия в числе литер, например:

01 MAJOR-B.

05 MINOR-H PICTURE IS X(10).

05 NEW-FIELD REDEFINES MINOR-H.

10 NUMBER-X PICTURE IS 9(4).

10 FILLER PICTURE IS X(6).

Переопределяемое данное может быть и групповым данным. Переопределение начинается с имени-данного и продолжается до тех пор, пока не будет достигнут номер-уровня, равный номеру-уровня переопределяемого данного. Допускается несколько переопределений одного и того же данного:

05 MINOR-G PICTURE IS A(3).

05 SMALL-G REDEFINES MINOR-G PICTURE IS 9(3).

05 FIELD-Z REDEFINES MINOR-G PICTURE IS X(3).

но переопределения переопределений запрещены. Следующая последовательность переопределений не верна:

05 MINOR-G PICTURE IS A(3).

05 SMALL-G REDEFINES MINOR-G PICTURE IS 9(3).

05 FIELD-Z REDEFINES SMALL-G PICTURE IS X(3).

(НЕВЕРНО!)

Фраза REDEFINES может быть использована для полной переорганизации поля памяти, выделенного под исходное данное, например:

01 RECORD-SPACE.

05 STORAGE-AREA.

```

10 ID-NUMBER          PICTURE IS 9(10).
10 PERSONS-NAME       PICTURE IS A(15).
05 TABLE-VALUES REDEFINES STORAGE-AREA.
10 FIRST-VALUE        PICTURE IS 9(8).
10 SECOND-VALUE.
    15 PART-A          PICTURE IS 9(4).
    15 PART-B          PICTURE IS 9(8).
10 FILLER              PICTURE IS X(5).
05 DIFFERENT-NAME REDEFINES STORAGE-AREA
    PICTURE IS X(25).

```

Существуют два ограничения на использование фразы REDEFINES. Ни в статье, описывающей переопределяющее-имя-данного, ни в подчиненных ей статьях не может присутствовать фраза VALUE IS. В статье, описывающей исходное имя-данного, фраза VALUE IS может присутствовать. При использовании варианта CORRESPONDING все переопределяющие-имена-данных игнорируются. Группы, упомянутые в операторах, использующих вариант CORRESPONDING, могут содержать фразы REDEFINES. Это не является ошибкой. Более того, эти группы сами могут быть подчинены данным, описанным с фразами REDEFINES.

Окончательно:

1) переопределяемые данные могут содержать подчиненные данные, которые получаются в результате переопределения:

```

05 TEST-A.
    10 ITEM-X PICTURE IS X.
    10 ITEM-Y PICTURE IS X.
    10 SMALL-A REDEFINES ITEM-Y PICTURE IS A.
05 NEW-A REDEFINES TEST-A PICTURE IS X(2).

```

2) переопределяющие-имена-данных (т. е. те, которые осуществляют переопределение) могут содержать подчиненные данные, которые в свою очередь являются переопределяющими:

```

05 TEST-A          PICTURE IS X(2).
05 NEW-A REDEFINES TEST-A.
    10 ITEM-G       PICTURE IS X.
    10 ITEM-H       PICTURE IS X.
    10 SMALL-C REDEFINES ITEM-X      PICTURE IS A.

```

3) за записью, содержащей фразы переопределения, могут следовать статьи переименования:

01 INPUT-RECORD.

05 ID-FIELD PICTURE IS X(10).

05 DIGITS-X REDEFINES ID-FIELD.

10 DIGITS-A PICTURE IS 9(2).

10 DIGITS-B PICTURE IS 9(2).

10 DIGITS-C PICTURE IS 9(2).

10 DIGITS-D PICTURE IS 9(2).

10 DIGITS-E PICTURE IS 9(2).

66 NEW-NAME RENAMES DIGITS-B THRU DIGITS-D.

Помните, что две рассмотренные фразы не изменяют длину записи. При подсчете числа литер, указываемого во фразе RECORD (В ЗАПИСИ), нужно использовать длину исходных данных.

Упражнения

1. Ниже приведены две статьи-описания-записи, которые могли бы встретиться в качестве описаний записей файла:

FD IN-FILE

LABEL RECORDS ARE STANDARD

DATA RECORDS ARE RECORD-ONE RECORD-TWO.

01 RECORD-ONE.

05 MAJOR-ONE.

10 MINOR-ONE.

15 ITEM-A PICTURE IS X(5).

15 ITEM-B PICTURE IS X(10).

10 MINOR-TWO.

15 ITEM-C PICTURE IS X(5).

15 ITEM-D PICTURE IS X(10).

05 MAJOR-TWO.

10 ITEM-E PICTURE IS X(6).

10 ITEM-F PICTURE IS X(9).

01 RECORD-TWO.

05 BIG-ONE.

10 FIRST-X PICTURE IS X(5).

10 SECOND-X PICTURE IS X(31).

05 BIG-TWO PICTURE IS X(9).

Предположим, что желательно, чтобы статья FD (ОФ) содержала только одну статью-описания-записи, а именно:

01 ONLY-THIS-REC PICTURE IS X(45).

и что используется оператор READ . . . INTO:

READ ONLY-THIS-REC RECORD INTO
TEST-RECORD AT END GO TO P-10.

Напишите статью-описания-записи для записи TEST-RECORD, которая позволяла бы задавать для этой записи и структуру, которую имеет запись RECORD-ONE, и структуру, которую имеет запись RECORD-TWO.

2. Имеется данное длиной в восемь буквенно-цифровых литер. Известно, что в поле этого данного располагается слово, состоящее точно из пяти литер. Однако не известно, как расположено это слово: оно может начинаться в любой из позиций 1, 2, 3 или 4. Напишите статью-описания-данного для этого данного и процедурный сегмент для помещения находящегося в нем слова в данное:

77 EXTRACTED-WORD PICTURE IS X(5).

Данное упражнение знакомит с возможностью произвольного расположения значения в пределах определенных границ и допускает некоторую гибкость при вводе данных в систему обработки информации.

3. Напишите статью-описания-данного, которая будет экономить место в секции рабочей-памяти путем объединения следующих записей в одной области:

01 STORAGE-X.

05 VENDOR-NUMBER PICTURE IS 9(5).

05 VENDOR-LOCATION.

10 V-NAME PICTURE IS A(15).

10 V-ADDRESS.

15 V-STREET PICTURE IS X(20).

15 V-CITY PICTURE IS A(15).

15 V-STATE PICTURE IS A(2).

05 S-CODE PICTURE IS X(3).

01 STORAGE-Y.

05 VENDOR-NUMBER PICTURE IS 9(5).

05 INVOICE-AMOUNT PICTURE IS 9(5)V99.

05 DISCOUNT-RATE PICTURE IS VP99.

01 STORAGE-Z.

05 VENDOR-NUMBER PICTURE IS 9(5).

05 DUE-DATE PICTURE IS 9(6).

05 S-CODE PICTURE IS X(3).

Память может быть сэкономлена потому, что одновременно в памяти будет храниться только одна из этих записей.

7.7. Межпрограммные связи

Иногда бывает удобно, находясь в некоторой КОБОЛ-программе, вызывать какую-либо другую полную КОБОЛ-программу. При решении больших и сложных задач лучше всего разбивать программу на отдельные логически самостоятельные части и поручать программирование этих частей разным программистам. Когда каждая из частей будет написана, проверена и отлажена, их можно будет объединить в одну большую программу с большей степенью уверенности в ее безошибочности, чем в случае, когда программа для решения поставленной задачи писалась бы как единое целое. Главная программа может вызывать каждую из подчиненных программ в требуемом порядке. После завершения выполнения подчиненной программы управление будет передано назад в вызывающую главную программу, которая вызовет затем следующую подчиненную программу, и так далее до тех пор, пока задача не будет полностью решена. Каждая из подчиненных программ является полной программой со всеми четырьмя разделами и со своими собственными именами файлов, записей и данных.

Программа, которая вызывает в память другую программу и запускает ее, называется *главной* программой; вызываемая подобным образом программа называется *подчиненной*. Главная программа может вызывать много различных подчиненных программ. Такая последовательность вызовов показана на рис. 7.2, где программа А по очереди вызывает программы X, Y и Z. Подчиненная программа в свою очередь может вызывать свои подчиненные программы, будучи по отношению к ним главной. Соответствующая последовательность управляющих вызовов показана на рис. 7.3. В этой последовательности подчиненная программа может быть главной по отношению к любому числу ее собственных подчиненных программ. Однако подчиненная программа никогда не может вызвать ни одну из программ, главных по отношению к ней в этой иерархии. Любая программа может вызвать любую другую программу, если только вызываемая программа никогда не вызывает вызывающую программу или любую другую программу, главную по отношению к вызывающей.

Вызовы программ

Для запуска подчиненной программы главная программа использует оператор CALL (ВЫЗВАТЬ). Общий формат простейшего оператора CALL имеет вид

CALL литерал

Главная программа:	
A	A-1. CALL "X".
	A-2. CALL "Y".
	A-3. CALL "Z".
	A-4. STOP RUN.
Подчиненная программа:	
X	X-1. ...
	X-2. ...
	X-3. EXIT PROGRAM.
Подчиненная программа:	
Y	Y-1. ...
	Y-2. ...
	Y-3. EXIT PROGRAM.
Подчиненная программа:	
Z	Z-1. ...
	Z-2. ...
	Z-3. EXIT PROGRAM.
Последовательность выполнения:	
A-1, X-1, X-2, X-3, A-2, Y-1, Y-2, Y-3, A-3, Z-1, Z-2, Z-3, A-4.	

Рис. 7.2. Связь между главной и подчиненными программами.

Главная программа:	
A	A-1. ...
	A-2. CALL "X".
	A-3. STOP RUN.
Подчиненная программа:	
X	X-1. CALL "Y".
	X-2. ...
	X-3. EXIT PROGRAM.
Подчиненная программа:	
Y	Y-1. CALL "Z".
	Y-2. CALL "W".
	Y-3. EXIT PROGRAM.
Подчиненная программа:	
Z	Z-1. ...
	Z-2. ...
	Z-3. EXIT PROGRAM.
Подчиненная программа:	
W	W-1. ...
	W-2. ...
	W-3. EXIT PROGRAM.
Последовательность выполнения:	
A-1, A-2, X-1, Y-1, Z-1, Z-2, Z-3, Y-2, W-1, W-2, W-3, Y-3, X-2, X-3, A-3.	

Рис. 7.3. Программная связь с помощью вложенных вызовов.

где литерал должен быть нечисловым и должен являться именем программы, указанным в параграфе PROGRAM-ID (ПРОГРАММА) подчиненной программы. Например:

```
CALL "PROG650"
CALL "TEST"
```

Подчиненная программа должна содержать предложение, возвращающее управление в главную программу. Таким предложением является предложение EXIT PROGRAM (ВЫЙТИ ИЗ ПРОГРАММЫ), которое должно быть единственным предложением параграфа, например:

```
FINAL-PARAGRAPH.
EXIT PROGRAM.
```

Параграф, содержащий предложение EXIT PROGRAM, может находиться в любом месте подчиненной программы. Он вовсе не обязательно должен быть последним параграфом, как было в случае предложения EXIT (ВЫЙТИ), описанного в предыдущей главе. Однако действует он точно так же, возвращая управление оператору главной программы, следующему за оператором CALL. Если программа, в которой написано предложение EXIT PROGRAM выполняется самостоятельно (а не будучи вызвана какой-либо другой программой), то это предложение игнорируется. В этом случае будет просто выполняться написанный вслед за ним параграф. В силу этого любая программа может быть подготовлена для работы в качестве подчиненной программы простым добавлением параграфа с предложением EXIT PROGRAM перед параграфом с оператором STOP RUN (ОСТАНОВИТЬ РАБОТУ). В этом случае программист должен обеспечить, чтобы в оставшейся части программы не было других операторов STOP RUN. Ниже приведен пример главной программы, вызывающей подчиненную программу:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.    SENIOR.
```

```
ENVIRONMENT DIVISION.
```

```
...
```

```
DATA DIVISION.
```

```
...
```

```
PROCEDURE DIVISION.
```

```
...
```


A-1.

READ IN-REC RECORD AT END GO TO A-10.
IF TEST-POSITION IS EQUAL TO "G" CALL "JUNIOR"
GO TO A-1.

...

IDENTIFICATION DIVISION.

PROGRAM-ID. JUNIOR.

ENVIRONMENT DIVISION.

...

DATA DIVISION.

...

PROCEDURE DIVISION.

B-1.

OPEN EXTEND NEW-FILE.
MOVE ZERO TO COUNT-X.

B-2.

WRITE SOME-RECCRD.
ADD 1 TO COUNT-X IF COUNT-X IS LESS THAN 75 GO
TO B-2.
CLOSE NEW-FILE.

B-3.

EXIT PROGRAM.

B-4.

STOP RUN.

Подчиненная программа, названная JUNIOR, может работать самостоятельно, используя параграф B-4 для прекращения работы, или может быть вызвана какой-либо другой программой, как в приведенном примере. Программа с именем SENIOR вызывает программу JUNIOR каждый раз, когда значением данного TEST-POSITION является буква G. Следовательно, файл с именем NEW-FILE будет открываться много раз. Имена файлов, записей, данных, параграфов и т. д. в обеих программах могут совпадать без нарушения правила однозначности имен, так как эти программы полностью независимы. Позаботиться придется только либо о фразах VALUE OF IDENTIFICATION (ЗНАЧЕНИЕ имя-реализации),

либо о командном языке, определяющем физические файлы, с тем, чтобы избежать противоречия между операциями над файлами в обеих программах.

При вызове подчиненной программы она загружается во внутреннюю память и выделяется место как для ее файловой области, так и для ее области рабочей-памяти. Будучи однажды вызвана, она остается в памяти в том же самом состоянии, в котором она находилась после выполнения оператора EXIT PROGRAM. Таким образом, любые значения, первоначально заданные с помощью фраз VALUE IS (ЗНАЧЕНИЕ) и менявшиеся впоследствии, при последующих вызовах уже не будут установлены в исходное состояние. Если бы вызывалась вторая, а затем и третья программа, как показано в примере на рис. 7.2, то доступная внутренняя память быстро оказалась бы заполненной и для следующей вызываемой программы уже не осталось бы в ней места. В этом случае используется оператор CANCEL (ОСВОБОДИТЬ) вида:

CANCEL {литерал} ...

который освобождает память, выделенную ранее для программ, имена которых указываются с помощью литералов, позволяя использовать ее для новых подчиненных программ. Следовательно, главную программу на рис. 7.2 лучше было бы написать следующим образом:

- A-1. CALL "X".
 CANCEL "X".
- A-2. CALL "Y".
 CANCEL "Y".
- A-3. CALL "Z".
- A-4. STOP RUN.

Когда встречается оператор STOP RUN, память, выделенная под все программы, освобождается и выполнение прекращается. Оператор CANCEL не препятствует повторному вызову той же самой программы, как, например, в случае предложения

IF TEST-POSITION IS EQUAL TO "G"
CALL "JUNIOR" CANCEL "JUNIOR".

При использовании такого предложения во время каждого нового вызова программы JUNIOR все значения будут снова находиться в исходном состоянии.

Совместное использование данных

При описании межпрограммных связей до сих пор предполагалось, что каждая программа имеет свою отдельную файловую область и отдельную область рабочей-памяти. При этом главная

программа не могла передавать значения данных подчиненной программе. Однако существует возможность, позволяющая подчиненной программе обрабатывать данные, принадлежащие главной программе. Сообщая подчиненной программе имена определенных данных, можно обеспечить их совместное использование подчиненной и главной программами. Это осуществляется путем установления соответствия между именами, используемыми в главной программе, и именами, используемыми для тех же данных в подчиненной программе. Таким образом, если имена данных в этих программах таковы:

в главной программе:

IN-REC, COUNT-OF-ERRORS, NUMBER-X

в подчиненной программе:

VALUE-RECORD, FREQUENCY-COUNT, NUMERICAL-VALUE

то между этими двумя списками имен можно установить связь так, что при помещении всех пробелов в данное VALUE-RECORD в подчиненной программе, очищаться будет данное, имеющее в главной программе имя IN-REC. Когда в подчиненной программе формируется значение данного FREQUENCY-COUNT, а затем происходит возврат в главную программу, в ней это значение будет доступно под именем COUNT-OF-ERRORS.

Соответствие между этими двумя списками имен и одними и теми же участками памяти устанавливается с помощью специальных статей, задаваемых в обеих программах. В главной программе такой список просто добавляется к оператору CALL, имеющему в этом случае формат

CALL литерал [USING идентификатор-1[идентификатор-2]..]

Например:

CALL "JUNIOR" USING IN-REC, COUNT-OF-ERRORS,
NUMBER-X

Как и всюду в КОБОЛе, запятые могут быть опущены.

Для завершения связи в подчиненной программе необходимо задать две статьи. Одна из них — это фраза USING (ИСПОЛЬЗУЯ), добавляемая к заголовку раздела процедур:

PROCEDURE DIVISION [USING идентификатор-1[идентификатор-2]..]

Например:

PROCEDURE DIVISION USING VALUE-RECORD,
FREQUENCY-COUNT, NUMERICAL-VALUE.

Положение каждого имени в обоих списках является существенным, так как оно служит для установления эквивалентности имен. Этих двух списков достаточно для установления эквивалентности имен, однако в разделе данных подчиненной программы необходимо задавать дополнительный набор статей. Подчиненной программе необходимо знать тип и размер значений совместно используемых данных. Эта информация задается в отдельной секции раздела данных, именуемой LINKAGE SECTION (СЕКЦИЯ СВЯЗИ). Секция LINKAGE должна следовать за секцией WORKING-STORAGE. Она включает статьи-описания-данного уровней 01 и 77, как и секция WORKING-STORAGE, но фразы VALUE IS для этих статей не допускаются. Секция LINKAGE не выделяет никакой памяти под данные, она просто переописывает область памяти, выделенную в главной программе, под совместно используемые данные. Использование статей описания в секции LINKAGE позволяет подчиненной программе изменять шаблоны совместно используемых данных при условии, что число литер в них остается неизменным. С этой точки зрения статьи в секции LINKAGE аналогичны фразам REDEFINES. С их помощью можно изменить структуру и категорию данных, но нельзя изменить их длину. Ниже приведен пример главной и подчиненной программ с совместно используемыми данными.

IDENTIFICATION DIVISION.
PROGRAM-ID. SENIOR.

...

DATA DIVISION.

FILE SECTION.

...

01 IN-REC PICTURE IS X(50).

WORKING-STORAGE SECTION.

77 NUMBER-X PICTURE IS 9(4) VALUE IS ZERO.

01 COUNT-OF-ERRORS PICTURE IS 9(5) VALUE IS ZERO.

...

PROCEDURE DIVISION.

...

A-1.

CALL "JUNIOR" USING IN-REC, COUNT-OF-ERRORS,
NUMBER-X.

...

IDENTIFICATION DIVISION.
PROGRAM-ID. JUNIOR.

...

DATA DIVISION.

...

WORKING-STORAGE SECTION.

01 JUNIORS-OWN-DATA-ITEM PICTURE IS 9(5).
 VALUE IS ZERO.

LINKAGE SECTION.

77 NUMERICAL-VALUE PICTURE IS 9(4).
01 VALUE-RECORD.
 05 JUNIOR-PART-ONE PICTURE IS X(10).
 05 FILLER PICTURE IS X(10).
 05 JUNIOR-PART-TWO PICTURE IS X(30).
01 FREQUENCY-COUNT PICTURE IS 9(5).

PROCEDURE DIVISION USING VALUE-RECORD,
FREQUENCY-COUNT, NUMERICAL-VALUE.

...

A-10.

EXIT PROGRAM.

Упражнения

I. Подготовьте список всех факторов, которые должен учитывать программист при создании и использовании файла данных. Этот список должен будет напоминать программисту о таких вещах, как имена полей, возможные диапазоны изменения величин, размеры записей, переопределения и переименования, используемые для операторов MOVE и IF, обнаружение ошибок, распреде-

ление внутренней памяти, назначение устройств, на которых будет установлен файл, и т. д. Такой обзорный список, помогающий в работе, следует сделать по всем предыдущим главам.

2. Предположим, что имеются две катушки магнитной ленты, одна из которых предназначена для хранения файла OLD-PERSONNEL-FILE, а другая — для хранения файла NEW-PERSONNEL-FILE. Программа будет обновлять файл OLD-PERSONNEL-FILE, копируя его в файл NEW-PERSONNEL-FILE с любыми необходимыми изменениями. При однократном цикле обновления тот файл, который сегодня назывался NEW-PERSONNEL-FILE, завтра будет называться OLD-PERSONNEL-FILE. Напишите подчиненную программу, которая при вращении катушек будет проверять, не перепутал ли их местами при установке оператор машины, и при обнаружении такой ошибки выдавать о ней сообщение.

3. Напишите КОБОЛ-программу для считывания файла, содержащего записи двух различных форматов.

Формат 1: (1) поле типа записи — двухцифровое поле, указывающее, что запись имеет тип 01; (2) поле, содержащее местонахождение завода, разбитое на поле города (буквенное, из двадцати пяти букв), поле штата из двух литер и поле почтового индекса (числовое, из пяти цифр).

Формат 2: (1) поле типа записи — двухцифровое поле, указывающее, что запись имеет тип 02; (2) поле описания продукции, разбитое на поле названия продукции (буквенно-цифровое, из тридцати пяти литер) и поле числового кода из пяти цифр; (3) числовое поле наличных запасов продукции из четырех цифр.

Файл состоит из записи типа 01, за которой следует неизвестное число записей типа 02. Последовательность записей типа 02 упорядочена по возрастанию кода продукции, причем один и тот же код могут иметь многие различные записи.

Программа должна проверять, что название города является буквенным, а почтовый индекс — числовым. Для каждого завода распечатайте:

- а) его местонахождение;
- б) список кодов продукции и общее количество наличных запасов продукции для каждого кода.

Глава 8. Таблицы

8.1. Общее представление о таблице

До сих пор структура данных, обрабатываемых вычислительной машиной с помощью КОБОЛ-программы, была представлена тремя уровнями.

Во-первых, это объединение литер в элементарные данные, чей размер, тип и значение определяются такими фразами, как PICTURE, VALUE IS, BLANK WHEN ZERO и т. д.

Во-вторых, эти данные организуются в виде иерархической структуры, подобной ветвям дерева, образуя групповые данные, из которых в свою очередь образуются более крупные групповые данные до тех пор, пока не будет составлена полная запись. (При этом существуют определенные ограничения, заключающиеся, например, в том, что данные или записи, имеющие номер уровня 77, должны быть элементарными.)

И, наконец, записи порождаются и запоминаются последовательно в виде файла данных на некотором физическом носителе, таком, как бобина с магнитной лентой или диск. При этом для хранения какого-либо большого файла может потребоваться несколько бобин или дисков, и, наоборот, несколько небольших файлов данных могут быть размещены на одной бобине или одном диске. Записи могут быть разной длины, но длина всех записей одного и того же типа одинакова.

Теперь будет введена другая структурная взаимосвязь между данными, хранящимися во внутренней памяти, а именно организация данных в виде таблиц. Таблица — это групповое данное, составленное из непрерывно расположенных данных, не снабженных отдельными именами. Данные, из которых составлена таблица, в свою очередь могут быть групповыми или элементарными и называются элементами таблицы. Например, таблица LIST-X могла бы быть следующим списком трехразрядных чисел:

833

927

401

228

При иерархической организации каждое данное в группе обязательно должно было бы иметь отдельное имя, например:

05 LIST-X.

10 FIRST-NUMBER	PICTURE IS 9(3).
10 SECOND-NUMBER	PICTURE IS 9(3).
10 THIRD-NUMBER	PICTURE IS 9(3).
10 FOURTH-NUMBER	PICTURE IS 9(3).

Преимуществом таблицы является то, что элементы таблицы не имеют отдельных имен и нет необходимости описывать каждый элемент по отдельности. В этом случае вместо описания, приведенного выше, достаточно, например, задать описание:

05 LIST-X PICTURE IS 9(3) OCCURS 4 TIMES.

Каждый элемент таблицы именуется с использованием порядкового номера, определяющего его позицию в таблице. Такая позиционная ссылка представляет собой имя таблицы, за которым следует указатель позиции, заключенный в круглые скобки. Например, ссылка на третий элемент таблицы может иметь вид

LIST-X(3)

Четыре элемента таблицы LIST-X имели бы имена LIST-X(1), LIST-X(2), LIST-X(3) и LIST-X(4). Вместо литерала, заключенного в круглые скобки, для обращения к элементам таблицы можно использовать имя данного, например:

LIST-X(LOCATION-Y)

Использование такой общей именуемой ссылки позволяет идентифицировать любой отдельный элемент просто с помощью присвоения значения данному LOCATION-Y. Таким образом, операторы

MOVE 3 TO LOCATION-Y.
DISPLAY LIST-X(LOCATION-Y).

выдали бы значение третьего элемента таблицы LIST-X, т. е. 401. Преимущество такой обобщенной нотации заключается в том, что ссылка может быть использована неоднократно, например:

MOVE 1 TO LOCATION-Y.
P-1.
DISPLAY LIST-X(LOCATION-Y).
ADD 1 TO LOCATION-Y.
IF LOCATION-Y IS NOT GREATER THAN 4
GO TO P-1.

Действия, описанные выше, эквивалентны следующему набору

операторов:

```
DISPLAY LIST-X(1).  
DISPLAY LIST-X(2).  
DISPLAY LIST-X(3).  
DISPLAY LIST-X(4).
```

В данном случае последний набор операторов представляет собой более короткий программный сегмент, но это было бы далеко не так для таблицы, состоящей из 200 элементов.

Общий формат для обращения к элементам таблицы таков:

имя-таблицы (индекс)

или

имя-таблицы (имя-индекса)

где имя-таблицы и индекс могут быть уточнены, в то время как имя-индекса уточнять нельзя.

В старых компиляторах требовалось, чтобы между именем-таблицы и левой скобкой был пробел, так как, согласно общему правилу КОБОЛа, имя может заканчиваться только пробелом или точкой. В современном стандарте правила пунктуации, касающиеся пробелов, стали менее строгими. Пробел перед или за круглой скобкой необязателен. Пробел справа от индекса или имени-индекса может быть опущен, так как имя может завершаться также правой круглой скобкой.

В последующих разделах будут описаны способы использования индексов и имен-индексов. Использование обращений с индексом является более простым и поэтому будет описано первым. При использовании имен-индексов порождается объектная программа, выполняющаяся быстрее. Однако при этом от программиста требуется большая тщательность, чтобы избежать ошибок. В конкретных ситуациях каждый из методов будет иметь свои преимущества и недостатки, и программист должен будет выбрать наиболее подходящий.

Выделение места во внутренней памяти для хранения таблицы осуществляется в соответствии с фразой OCCURS (ПОВТОРЯЕТСЯ), простейшая форма которой имеет вид

OCCURS целое TIMES

где целое должно быть положительным. Фраза OCCURS будет иметь более сложный вид при использовании имен-индексов. Фраза OCCURS не может быть использована в статьях-описания-данного, имеющих номер-уровня 01, 66 или 77. Ее можно употреблять только внутри формата записи. Примеры описаний таблиц приводятся ниже:

01 RECORD-A.

05 LIST-X	OCCURS 4 TIMES	PICTURE IS 9(3).
05 STATES-X	OCCURS 50 TIMES.	
10 POPULATION-OF-STATE		PICTURE IS 9(10).
10 NAME-OF-STATE		PICTURE IS A(15)
		JUSTIFIED RIGHT.
05 BUILDING-FLOORS.		
10 NAME-OF-OCCUPANT		OCCURS 10 TIMES
		PICTURE IS X(30).
10 NUMBER-OF-EMPLOYEES		OCCURS 10 TIMES
		PICTURE IS 9(5).

Использование фразы VALUE IS (ЗНАЧЕНИЕ) для задания значения не допускается ни в статье-описания-данного, содержащей

Размер статьи (в литерках)	Возможное значение статьи	Фактический адрес памяти	Относительный адрес памяти	Порядковый номер
3	833	4030 *	0	1
3	927	4033	3	2
3	401	4036	6	3
3	228	4039	9	4
10	0003383709	4042 *	0	1
15	ALABAMA	4052 *	0	1
10	0000989919	4067	25	2
15	MAINE	4077	25	2
...
10	0009698097	5267	1225	50
15	OHIO	5277	1225	50
30	JONES & CO.	5292 *	0	1
30	SMITH, LTD.	5322	30	2
...
30	ROBERTS AND SON	5562	270	10
5	00115	5592 *	0	1
5	00232	5597	5	2
...
5	00089	5637	45	10

* Означает начальный адрес памяти

Рис. 8.1. Структура записи RECORD-A.

фразу OCCURS, ни в подчиненных ей статьях. Это ограничение можно обойти путем использования фразы REDEFINES (ПЕРЕОПРЕДЕЛЯЕТ). Если фраза REDEFINES не употребляется, то значения элементов могут быть определены только с помощью чтения в записи, пересылкой требуемого значения или присваиванием результата некоторого вычисления. Возможные значения записи

RECORD-A приведены на рис. 8.1, показывающем структурную взаимосвязь данных в таблицах, соответствующие им абсолютные адреса памяти, относительные адреса памяти и порядковые номера. Определить порядковый номер данного несложно, так как это относительная величина, отсчитываемая начиная с единицы, но абсолютные адреса памяти обычно не известны программисту. Абсолютный адрес памяти — это адрес ячейки внутренней памяти, в которой хранится значение отдельного элемента. Относительный адрес памяти — это смещение относительно начального адреса таблицы. Он может быть вычислен по порядковому номеру. В примере на рис. 8.1 относительный адрес вычисляется по формуле

относительный-адрес = (порядковый-номер — 1) * размер-данного

где размер-данного задается для каждого элемента фразой PICTURE. Смещение для первого элемента всегда равно нулю. Абсолютный адрес памяти для данного может быть получен прибавлением относительного адреса к начальному адресу. При выполнении рабочей программы абсолютные адреса вычисляются именно таким образом.

В рассматриваемом примере RECORD-A это не имя-таблицы. Это имя группового данного, размер которого равен 1612 позициям, включая 12 позиций таблицы LIST-X, 1250 позиций таблицы STATES-X, 300 позиций таблицы NAME-OF-OCCUPANT и 50 позиций таблицы NUMBER-OF-EMPLOYEES. Подчиненное групповое данное BUILDING-FLOORS — это также не таблица. Каждый раз при использовании имен подчиненных элементов таблицы STATES-X необходимо указывать индекс или имя-индекса. Обращение к элементам таблиц в записи RECORD-A осуществляется следующим образом:

LIST-X (LOCATION-Y)
STATES-X (SUBSCRIPT-1)
POPULATION-OF-STATE (SUBSCRIPT-2)
NAME-OF-STATE (SUBSCRIPT-3)
NAME-OF-OCCUPANT (NUMBER-X)
NUMBER-OF-EMPLOYEES (FLOOR-NUMBER)

Прежде чем выполнить определенное обращение к какому-либо элементу таблицы, необходимо присвоить значение соответствующему индексу или имени-индекса. (Имена, использованные в приведенных выше примерах в качестве индексов, выбраны произвольно и могут определяться программистом.) Элементами таблицы STATES-X являются групповые данные. Обращаться к ним можно либо как к групповым элементам (STATES-X (AB)), либо как к элементарным элементам (POPULATION-OF-STATE (SUB)). Размер элементов таблицы STATES-X достаточен для хранения

25 литер, размер элементов с именем POPULATION-OF-STATE составляет 10 позиций. В нашем примере элемент STATES-X(1) имеет значение:

0003383709

ALABAMA

элемент POPULATION-OF-STATE(1) —

0003383709

и элемент NAME-OF-STATE(1) —

ALABAMA

которое сдвинуто вправо в соответствии со статьей-описания. Имя-таблицы может быть уточнено

STATES-X OF RECORD-A (AB)

но следует заметить, что индекс или имя-индекса должны следовать за полным уточнением. Нижеследующий порядок недопустим:

STATES-X(AB) OF RECORD-A (НЕВЕРНО!)

8.2. Использование индексов

Существует два способа определения положения элемента в таблице; один использует индексные значения, а другой — имена-индексов. В этом разделе будет рассмотрен первый способ. Индекс может быть либо числовым литералом, либо именем элементарного данного, имеющего целое положительное значение. Примерами индексированных имен-таблиц являются:

LIST-X(4)

STATES-X (AB)

NAME-OF-OCCUPANT (SUB-X)

При использовании индексов не допускаются ни отрицательные значения, ни нуль. Наименьшим допустимым значением является 1, а наибольшим допустимым значением является значение, определенное во фразе OCCURS для данной таблицы. Имя-данного, используемого в качестве индекса, может быть уточнено, но оно не может быть в свою очередь индексировано, и в качестве индекса не может быть использовано арифметическое выражение. Однако значение индекса может быть определено любым образом в отдельном операторе, после чего его можно использовать вместе с именем-таблицы, например:

COMPUTE SUB-X = 3.0 * VALUE-A + 2.0.

MOVE STATES-X (SUB-X) TO RESULT-X.

Значение индекса представляет собой порядковый номер элемента в таблице. Вычислительная машина вычисляет относительный адрес памяти, вычитая единицу из порядкового номера и умножая эту разность на размер элемента таблицы. Для получения абсолютного адреса требуемого элемента результат умножения добавляется к начальному адресу таблицы. Если в качестве индекса используется числовой литерал, то такое вычисление проводится только один раз (во время компиляции). Если в качестве индекса используется имя-данного, то вычисление адреса осуществляется во время выполнения и повторяется при каждом обращении к элементу. Это вычисление может включать несколько шагов и требует определенных затрат машинного времени, что является недостатком этого способа. Как с точки зрения экономии места при записи КОБОЛ-программы в памяти при ее работе, так и с точки зрения уменьшения времени работы, лучше использовать оператор

MOVE LIST-X(3) TO ANSWER-X.

чем операторы

MOVE 3 TO LOG-Y.

MOVE LIST-X (LOG-Y) TO ANSWER-X.

В следующем примере будет выполнено четыре отдельных вычисления абсолютного адреса памяти:

COMPUTE SUB-X = 3.0 * VALUE-A + 2.0.

ADD INCREASE-X TO POPULATION-OF-STATE(SUB-X).

IF POPULATION-OF-STATE (SUB-X) IS GREATER THAN
900000 SUBTRACT ADJUST-X FROM
POPULATION-OF-STATE (SUB-X).

MOVE POPULATION-OF-STATE (SUB-X) TO SPECIAL-X.

С другой стороны, использование индексов предоставляет программисту простое средство для выполнения многих повторяющихся операций. Например, общая операция над каждым из многих полей записи может быть выполнена следующим образом:

77 SUB-A

PICTURE IS 9(3).

01 RECORD-B.

05 REPEATED-FIELD

OCCURS 50 TIMES

PICTURE IS 9(5).

.

.

.

READ RECORD-FILE RECORD INTO RECORD-B
AT END STOP RUN.

MOVE 1 TO SUB-A.

LOOP-A.

ADD 100,0 TO REPEATED-FIELD (SUB-A).

ADD 1 TO SUB-A.

IF SUB-A IS NOT GREATER THAN 50

GO TO LOOP-A.

Элементами таблиц могут быть групповые данные. Такую таблицу можно использовать для хранения нескольких связанных данных в одном элементе. При поиске в таблице конкретного значения одного данного будет возникать значение индекса, которое могло бы быть использовано для работы со связанными данными. Рассмотрим пример:

77 SUB-GROUP VALUE IS 1 PICTURE IS 9(3).

01 RECORD-C.

05 GROUP-ENTRIES OCCURS 500 TIMES.

10 NAME-OF-PRODUCT PICTURE IS X(25).

10 COST-OF-PRODUCT PICTURE IS 9(5).

.

.

.

P-1.

IF COST-OF-PRODUCT (SUB-GROUP) IS EQUAL TO 1500
GO TO P-2.

ADD 1 TO SUB-GROUP.

GO TO P-1.

P-2.

MOVE NAME-OF-PRODUCT (SUB-GROUP) TO PRINT-LINE.

Этот пример имеет некоторые недостатки. Хотя начальное значение индекса SUB-GROUP может быть установлено с помощью фразы VALUE, этим лучше не пользоваться, так как установка значения 1 происходит только при начальной загрузке программы и во время работы программы это значение может быть изменено каким-либо оператором, предшествующим P-1. Лучше присваивать начальное значение в начале каждого цикла. Другим недостатком является то, что эту процедуру нельзя использовать в случае, когда среди значений COST-OF-PRODUCT (SUB-GROUP) нет равного 01500. Здесь нет выхода из цикла в случае, если условие не выполняется. Такой выход следует всегда предусматривать, несмотря на уверенность программиста, что условие обязательно будет выполнено. И, наконец, процедура выбирает только то зна-

чение NAME-OF-PRODUCT (SUB-GROUP), которое связано с первым вхождением цены 01500. Повторные вхождения ее в таблицу игнорируются.

Связанные статьи не обязательно должны быть записаны в одной и той же таблице при условии, что они находятся в одинаковых относительных позициях, например:

01 RECORD-D.

05 LIST-A OCCURS 5 TIMES PICTURE IS 9(3).

01 RECORD-E.

05 LIST-B OCCURS 5 TIMES PICTURE IS 9(5).

MOVE 1 TO SUB-GROUP.

P-1.

ADD LIST-A (SUB-GROUP) TO LIST-B (SUB-GROUP).

ADD 1 TO SUB-GROUP.

IF SUB-GROUP IS NOT GREATER THAN 5

GO TO P-1.

Другой путь выполнения этого же сложения таблиц состоит в использовании иерархического описания и уточненных имен, а именно:

01 RECORD-D.

01 RECORD-E.

05 A PIC IS 9(3).

05 A PIC IS 9(3).

05 B PIC IS 9(3).

05 B PIC IS 9(3).

05 C PIC IS 9(3).

05 C PIC IS 9(3).

05 D PIC IS 9(3).

05 D PIC IS 9(3).

05 E PIC IS 9(3).

05 E PIC IS 9(3).

с последующим использованием оператора

ADD CORRESPONDING RECORD-D TO RECORD-E.

Однако вариант CORRESPONDING (СООТВЕТСТВУЮЩИЙ) не может быть использован для замены программного цикла в параграфе P-1, т. е. оператор

ADD CORRESPONDING LIST-A TO LIST-B. (НЕВЕРНО!)

запрещен. На самом деле не допустимо упоминать имя-таблицы без индекса или имени-индекса. Тем не менее вариант CORRESPONDING может быть использован для элементов таблицы в случае, когда эти элементы являются групповыми данными. Для следующих двух наборов статей-описания-данных:

01 GROUP-A.

05 FIELD-X

OCCURS 10 TIMES.

10 A PIC IS X.

10 B PIC IS X.

10 C PIC IS X.

10 D PIC IS X.

01 GROUP-B.

05 FIELD-Y

OCCURS 10 TIMES.

10 A PIC IS X.

10 Z PIC IS X.

10 Y PIC IS X.

10 B PIC IS X.

допустим оператор такого типа:

MOVE CORRESPONDING FIELD-X (SUB-A) TO
FIELD-Y (SUB-B).

Соответствующие элементы будут идентифицироваться с помощью индексов SUB-A и SUB-B. Оператор MOVE будет применен к данным с именами A и B.

Уже упоминалось, что фраза VALUE IS не может появиться в статье-описания-данного, содержащей фразу OCCURS. Существует два пути обойти это ограничение в задании начальных значений элементов таблицы. Первый применяется в простых случаях присвоения одного и того же значения всем элементам таблицы и основывается на том, что фраза VALUE может появиться в описании группового данного и относится ко всем элементарным данным соответствующей группы, например:

05 GROUP-ITEM VALUE IS ZERO.

10 ELEMENT-X OCCURS 10 TIMES PICTURE IS 9(5).

Более общий путь задания начальных значений состоит в использовании обычного иерархического определения, допускающего употребление фразы VALUE, с последующим переопределением этой группы как таблицы, например:

05 GROUP-ITEM.

10 A VALUE IS 25.0 PICTURE IS 9(3)V9.

10 B VALUE IS 30.0 PICTURE IS 9(3)V9.

10 C VALUE IS 35.0 PICTURE IS 9(3)V9.

05 ARRAY-X REDEFINES GROUP-ITEM

OCCURS 3 TIMES PICTURE IS 9(3)V9.

Фраза REDEFINES должна следовать сразу же за именем-таблицы. Этот метод часто используется для организации таблиц ссылок в области рабочей-памяти.

Упражнения

1. Нарисуйте схему структуры данных, заданной следующим описанием:

01 ENTIRE-LIST.

05 LINE-ENTRY OCCURS 10 TIMES PICTURE IS A(10).

05 ITEM-X OCCURS 10 TIMES PICTURE IS 9(3).

2. Напишите статью-описания-данного во внутренней памяти для страницы грессбуха, содержащей в каждой строке номер счета, название счета, последний вклад и остаток для одной сотни счетов.

3. Используя индексы, напишите программный сегмент для определения суммы всех остатков предыдущего упражнения.

4. Используя индексы, напишите полную КОБОЛ-программу для чтения записей, состоящих из данных, именуемых инвентарными-номераами и имеющих шаблон PICTURE IS 9(5), и заполнителей FILLER X(55). Распечатывайте инвентарные-номера по двадцать в строку до тех пор, пока весь файл не будет прочитан и распечатан. Возможно, что файл не будет содержать целое число раз по двадцать записей, так что заполните остаток последней строки пробелами.

8.3. Таблицы таблиц

Элементы таблицы могут быть не только элементарные данные, но и групповые данные. Таблица определяется как групповое данное, состоящее из подряд расположенных элементов с одним и тем же именем. Объединяя эти два утверждения, можно предположить, что элементом таблицы в свою очередь может быть таблица. В КОБОЛе это верно вплоть до двух уровней вложенности таблиц в таблицы. Рассмотрим следующую статью-описания-данного:

01 RECORD-Z.

05 ITEM-X OCCURS 2 TIMES.

10 ITEM-Y OCCURS 3 TIMES PICTURE IS X(2).

Это описание определяет групповое данное, состоящее из шести элементарных данных (каждое из двух литер), организованное следующим образом:

RECORD-Z (групповое данное)

ITEM-X(1): ITEM-Y(1, 1) ITEM-Y(1, 2) ITEM-Y(1, 3)

ITEM-X(2): ITEM-Y(2, 1) ITEM-Y(2, 2) ITEM-Y(2, 3)

Если бы имела место следующая передача данных:

MOVE "ABCDEFGHijkl" TO RECORD-Z

то значения различных элементов были бы таковы:

ITEM-X(1) ABCDEF

ITEM-X(2) GHIJKL

ITEM-Y(1, 1) AB
ITEM-Y(2, 1) GH

ITEM-Y(1, 2) CD
ITEM-Y(2, 2) IJ

ITEM-Y(1, 3) EF
ITEM-Y(2, 3) KL

Для идентификации этих шести элементов требуется две позиционные ссылки (которые могут быть либо индексами, либо именами-индексов). Эти позиционные ссылки могут быть разделены запятыми с использованием правила, по которому пробел обязательно должен следовать за запятой и может предшествовать ей. (Эти запятые, как и все запятые в КОБОЛе, не обязательны. Здесь они используются для более четкого разделения ссылок.) Каждый из элементов может быть элементарным данным (как в приведенном выше примере), групповым данным или другой таблицей. Последнее будет тем пределом, который допустим в КОБОЛе: элементами третьей таблицы могут быть только элементарные или групповые данные, но не таблицы. В качестве примера двух уровней вложенности можно привести следующее описание:

01 RECORD-Y.

05 ITEM-X OCCURS 2 TIMES.

10 ITEM-Y OCCURS 3 TIMES.

15 ITEM-Z OCCURS 2 TIMES.

20 ENTRY-A PICTURE IS X(3).

20 ENTRY-B PICTURE IS X(5).

Предположив, что I, J и K — это имена-данных, используемых в качестве индексов, можно представить различные уровни группового данного RECORD-Y таким образом:

ITEM-X(I)	для	I	от 1 до 2
ITEM-Y(I, J)	для	I	от 1 до 2 и для
		J	от 1 до 3
ITEM-Z(I, J, K)	для	I	от 1 до 2 и для
		J	от 1 до 3 и для
		K	от 1 до 2

Так как статьи ENTRY-A и ENTRY-B подчинены статье ITEM-Z, обращаться к ним следует с тем же числом индексов, например:

ENTRY-A (2, 1, 2)

Заметим, что порядок индексов или имен-индексов внутри круглых скобок слева направо соответствует порядку старшинства таблиц. На рис. 8.2 показана структурная связь между элементами в таблице RECORD-Y. С помощью фраз REDEFINES и RENAMES можно добиться комбинации элементарных, групповых и табличных элементов. Однако программист не может использовать фразу REDEFINES ни для имени данного, с которым связана фраза

OCCURS, ни для имени данного, подчиненного данному, в описании которого встречается фраза OCCURS. Тем не менее фраза REDEFINES может присутствовать в описании данного, старшего по отношению к данному, в описании которого встречается фраза OCCURS, при условии, что размеры таблиц, входящих в это дан-

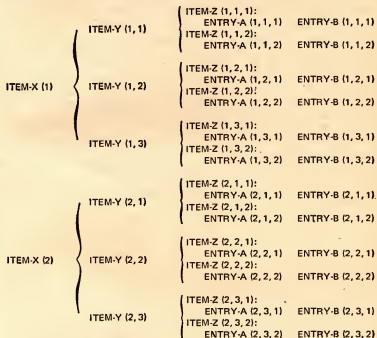


Рис. 8.2. Структура записи RECORD-Y.

ное, не меняются. (См. описание фразы DEPENDING ON (В ЗАВИСИМОСТИ ОТ) в разд. 8.6.) В качестве примера возможной комбинации данных рассмотрим описание:

01 RECORD-X.

05 ALPHA.

10 A OCCURS 3 TIMES.

15 B.

20 C OCCURS 2 TIMES

25 D PICTURE IS X(3).

25 E OCCURS 4 TIMES PICTURE IS X.

20 F OCCURS 3 TIMES PICTURE IS X(2).

20 G PICTURE IS X(4).

05 XRAY REDEFINES ALPHA.

10 Z OCCURS 6 TIMES PICTURE IS X(12).

66 ABLE RENAMES ALPHA.

В статье-описания-данного для имени-данного ALPHA, упомянутого во фразе RENAMES, не должно содержаться фразы OCCURS. Этой фразы не должно быть и ни в каких статьях, старших по отношению к рассматриваемой статье.

Метод индексирования можно использовать для чтения величин и подсчета их сумм во внутренней таблице. Предположим, что в трехцифровом коде номера-счета, например 312, первая цифра используется для указания отдела, к которому относится счет, вторая цифра используется для указания типа счета (командировка, оклад или разное), третья цифра используется для идентификации темы. Пусть для простоты имеется всего четыре отдела, восемь различных типов счетов и шесть тем. Отчет о расходах должен состоять из номера счета и потраченной суммы. Всего возможных номеров счетов будет $4 \times 8 \times 6$, т. е. 192. Следующая ниже программа будет считывать суммы, потраченные за последний месяц, и накапливать их в соответствии с номерами счетов в таблице:

•
•
•

FD CARD-IN-FILE

LABEL RECORD IS STANDARD.

01 CARD-RECORD.

05 ACCOUNT-NUMBER.

10 DEPARTMENT-X PICTURE IS 9.

10 ACCOUNT-TYPE PICTURE IS 9.

10 PROJECT-X PICTURE IS 9.

05 AMOUNT-EXPENDED PICTURE IS 999V99.

•
•
•

WORKING-STORAGE SECTION.

01 TABLE-OF-ACCOUNTS VALUE IS ZERO.

05 BY-DEPARTMENT OCCURS 4 TIMES.

10 BY-TYPE OCCURS 8 TIMES.

15 TOTAL-AMOUNT OCCURS 6 TIMES
PICTURE IS 9(10)V99.

.
.
.

P-1.

READ CARD-IN-FILE RECORD AT END GO TO ANOTHER-PARAGRAPH.

ADD AMOUNT-EXPENDED TO

TOTAL-AMOUNT (DEPARTMENT-X, ACCOUNT-TYPE, PROJECT-X).

GO TO P-1.

После того как будет прочитан весь файл расходов, можно будет подсчитывать суммы по различным индексам. Следующий сегмент будет распечатывать общие затраты каждого отдела (в предположении, что все используемые данные описаны необходимым образом):

.
.
.

MOVE 1 TO SUB-A.

P-2.

MOVE ZERO TO DEPARTMENT-TOTAL.

MOVE 1 TO SUB-B.

P-3.

MOVE 1 TO SUB-C.

P-4.

ADD TOTAL-AMOUNT (SUB-A, SUB-B, SUB-C) TO DEPARTMENT-TOTAL.

ADD 1 TO SUB-C.

IF SUB-C IS NOT GREATER THAN 6 GO TO P-4.

ADD 1 TO SUB-B.

IF SUB-B IS NOT GREATER THAN 8 GO TO P-3.

P-5.

DISPLAY "DEPARTMENT NO." SUB-A

"SPENT A TOTAL OF" DEPARTMENT-TOTAL.

ADD 1 TO SUB-A.

IF SUB-A IS NOT GREATER THAN 4 GO TO P-2.

Упражнения

1. В компании имеется 1000 служащих, на каждого из которых заведена персональная запись. Все записи упорядочены в алфавитном порядке и хранятся на магнитной ленте. Длина каждой записи 600 литер. В позициях с 11 по 16 включительно хранится шестизначный номер, представляющий собой дату рождения, так что первые две цифры указывают год, следующие две цифры — месяц и последние две цифры — день рождения. Напишите программу подготовки списка числа людей, имеющих один и тот же день рождения, независимо от года рождения. Таким образом, номера 250623 и 340623 означают один и тот же день рождения, хотя возраст соответствующих людей различен. Всего может существовать не более 366 различных дат рождения, но в данной компании могут присутствовать не все возможности. Список дат (только месяц и день) и числа людей, родившихся в этот день, должен содержать только даты, в которые родился хотя бы один сотрудник компании.

2. Каждый раз, когда рабочий в цеху выполняет определенное задание, он пробивает на карте номер задания и затраченное время в минутах. Для двухчасового задания на карте будет пробито 120. Никакое задание не может выполняться более восьми часов подряд. Существует в точности 100 различных номеров заданий, но что это за номера, не известно. Номера заданий имеют шаблон X(8), но их значения не будут известны до тех пор, пока карта задания не будет прочтана. Напишите программу для чтения карт заданий в случайном порядке (в том порядке, в котором они составлялись в течение месяца) и подсчета суммарного времени, затраченного на каждое задание в течение месяца. Не сортируйте номера заданий для объединения одинаковых. Читайте их в том порядке, в котором они поступают. Распечатывайте итоговые суммы вместе с номерами заданий.

3. Имеется 500 автомобилей, снабженных номерами от 001 до 500, у которых проверяется длина пробега шин. Каждый раз, когда покрышка стирается на одну шестнадцатую дюйма, шина заменяется и расстояние, пройденное шиной, записывается. Предполагается, что у различных автомобилей в парке будет различный срок службы шин, так что каждая запись о шине содержит наряду с пройденным расстоянием и номер соответствующего автомобиля. Напишите программу считывания всех записей длины пробега

шин (для 50 000 шин, изношенных в течение проверки) и распечатайте среднюю длину пробега шин для каждого автомобиля. (Конечно, число замен шин у каждого автомобиля будет свое.)

8.4. Использование имен-индексов

Как уже отмечалось, таблица — это список элементов с общим именем-таблицы. Каждый элемент может быть элементарным или групповым данным или таблицей. Обращение к отдельному элементу осуществляется с помощью индекса или посредством имени-индекса. Значением индекса независимо от того, является ли он числовым литералом или именем-данного, всегда служит целое положительное число. Переход к абсолютному адресу ячейки, содержащей элемент во внутренней памяти, всегда осуществляется при каждом использовании элемента с индексом. Это требует вычисления абсолютного адреса посредством умножения порядкового номера (уменьшенного на единицу) на смещение и добавления результата к базовому адресу. Например, если бы базовый адрес равнялся 4030 и каждый элемент содержал три стандартных литеры, то абсолютный адрес четвертого элемента таблицы имел бы значение

$$(4 - 1) * 3 + 4030 = 4039$$

Метод, использующий имя-индекса для идентификации элемента, похож на предыдущий метод, но значением имени-индекса является не порядковый номер элемента, а относительный адрес памяти. И именно в этом различии заключается главное преимущество использования имени-индекса вместо индекса. Так как значением имени-индекса является уже относительный адрес, то вычисление абсолютного адреса заключается просто в добавлении относительного адреса к базовому адресу таблицы. В приведенном выше примере для порядкового номера, равного 4, значение имени-индекса, связанного с таблицей, равнялось бы 9. Вычисление абсолютного адреса элемента свелось бы к сложению:

$$9 + 4030 = 4039$$

При этом число вычислений по сравнению с использованием индекса уменьшится по крайней мере в четыре раза. Кроме того, при этом уменьшится размер рабочей программы за счет исключения лишних команд. Для того чтобы воспользоваться этим преимуществом, программист должен быть очень осторожен при использовании имени-индекса, потому что, в то время как значение индекса может ссылаться на элементы, имеющие одинаковое положение в любой таблице, значение имени-индекса может относиться только к одной таблице, его «родительской» таблице. Относитель-

ный адрес, скажем, четвертого элемента одной таблицы может быть никак не связан с относительным адресом четвертого элемента другой таблицы (за исключением некоторых простых случаев, которые вряд ли будут иметь место для различных вычислительных машин).

Имя-индекса — это новый тип данных и его расположение в памяти и вид определяются авторами компиляторов, а не программистом, а именно:

1) для имени-индекса место в файловой области или в области рабочей-памяти не выделяется и статья-описания-данного не нужна;

2) изменить имя-индекса можно только с помощью трех операторов SET (УСТАНОВИТЬ), SEARCH (ИСКАТЬ) и PERFORM (ВЫПОЛНИТЬ) и никак иначе.

Оператор SET будет описан в этом разделе, оператор SEARCH — в следующем разделе, а оператор PERFORM — в следующей главе. Но если INDEX-A это имя-индекса, то все следующие операторы не верны:

MOVE 1 TO INDEX-A. (НЕВЕРНО!)

ADD 5 TO INDEX-A. (НЕВЕРНО!)

MOVE LOC-Y TO INDEX-A. (НЕВЕРНО!)

Но прежде, чем имени-индекса можно будет присвоить какое-либо значение, оно должно быть связано с конкретной «родительской» таблицей. Имя-индекса записывается как и любое имя-данного в КОБОЛе, однако его нельзя уточнять и оно не может использоваться с индексом или именем-индекса. Связь с именем-таблицы устанавливается с помощью добавления к фразе OCCURS:

OCCURS целое TIMES

[INDEXED BY имя-индекса-1 [имя-индекса-2] ...]

причем с именем-таблицы может быть связано одно или более имен-индексов.

Только в том случае, если имя-таблицы описано таким образом, его можно использовать вместе с именем-индекса:

05 ELEMENT-Z OCCURS 20 TIMES

INDEXED BY INDEX-G PICTURE IS X.

а затем

MOVE "M" TO ELEMENT-Z (INDEX-G).

Правила, введенные для таблиц, сохраняются и при использовании имен-индексов: допускаются вложенные таблицы (пробелы или запятые между именами-индексов и т. д.). К элементу таблицы в одной и той же программе можно обращаться как с помощью индекса, так и с помощью имени-индекса. Очевидно, что обращение

к элементу LIST-X(3) производится с помощью индекса, так как для этого используется числовой литерал. В случае обращения LIST-X(G) без рассмотрения статьи-описания-данного невозможно сказать, используется ли индекс или имя-индекса. Этим анализом, конечно, занимается компилятор, так что программа не спутает эти случаи. Для того чтобы помочь программисту проследить за этим различием, полезно использовать для индексов и имен-индексов отличающиеся приставки, например: SUB — для индексов и INDEX — для имен-индексов¹⁾. В этом нет никакого другого смысла, кроме введения мнемоники, позволяющей избежать ошибок.

Могут возникнуть ситуации, при которых программисту необходимо будет запомнить значение имени-индекса, скажем, в качестве точки отсчета для дальнейших вычислений или для установки границ в какой-либо процедуре поиска. Есть возможность определить элементарное данное, предназначенное для хранения только адресных величин. Это делается с помощью фразы

USAGE IS INDEX

которая используется в статье-описания-данного для задания еще одного нового типа данных, называемого индексное-данное. Примером могла бы служить следующая статья:

77 STORED-VALUE USAGE IS INDEX.

Фраза USAGE может быть написана на любом уровне. Когда она встречается на групповом уровне, индексными-данными являются только элементарные данные этой группы, а группа в целом не является индексным-данным. Индексное-данное не является именем-индекса, но оно может быть использовано для индексирования в том случае, когда оно содержит значение имени-индекса соответствующей таблицы. Значениями индексного-данного являются относительные адреса, а не порядковые номера, и оно может быть изменено только с помощью операторов SET и SEARCH. В частности, индексное-данное не может быть операндом в операторе MOVE. Размер индексного-данного определяется авторами компилятора, а соответствующая статья-описания-данного может содержать только фразу USAGE IS INDEX. Подчеркнем, что в этой статье не может быть фраз PICTURE, VALUE IS, JUSTIFIED RIGHT или BLANK WHEN ZERO.

При чтении текста программы следует отличать имена-индексов от индексных-данных. Имя-индекса всегда связано с конкретной

¹⁾ В английском описании КОБОЛа индекс называется subscript, а имя-индекса — index-name. Отсюда и предлагаемая мнемоника. При использовании русской мнемоники для образования имен можно выбрать другие мнемонические приставки.— *Прим. перев.*

таблицей с помощью фразы INDEXED BY имя-индекса. Для имени-индекса не должно быть статьи-описания-данного, так как место его размещения и вид определяются авторами компилятора. С другой стороны, индексное-данное не связано ни с каким именем-таблицы и для него необходима статья-описания-данного с фразой USAGE IS INDEX. Ему не придается никакого шаблона, так как его размер и вид определяются авторами компилятора. Основное назначение индексного-данного состоит во временном хранении относительного адреса, являющегося значением некоторого имени-индекса. Индексное-данное можно использовать для индексации имени-таблицы, но это приведет к правильному результату только в том случае, когда оно хранит значение имени-индекса, связанного с данной таблицей.

Изменить значения имен-индексов и индексных-данных можно только с помощью глаголов SET, SEARCH и PERFORM (причем последний используется только для имен-индексов). Первый из этих глаголов SET используется для начального задания и изменения значений имен-индексов. Кроме того, его можно применять для передачи значений между именами-индексов, индексными-данными и другими данными. Существует два формата оператора SET.

Формат 1:

$$\text{SET} \left\{ \begin{array}{l} \text{имя-индекса-1} \quad [\text{имя-индекса-2}] \dots \\ \text{индексное-данное-1} \quad [\text{индексное-данное-2}] \dots \\ \text{идентификатор-1} \quad [\text{идентификатор-2}] \dots \end{array} \right\}$$

$$\text{TO} \left\{ \begin{array}{l} \text{целое} \\ \text{имя-индекса-3} \\ \text{индексное-данное-3} \\ \text{идентификатор-3} \end{array} \right\}$$

Примеры:

SET INDEX-A TO 1.

SET INDEX-A INDEX-B TO FIRST-VALUE.

SET INDEX-DATA-ITEM-A TO INDEX-A.

Имена-индексов должны быть связаны с определенным именем-таблицы с помощью фразы INDEXED BY. В описаниях индексных-данных должна присутствовать фраза USAGE IS INDEX, а идентификаторы должны быть элементарными целыми данными. Так как в оператор SET могут входить данные разных видов, существуют определенные ограничения на допустимые их комбинации:

1. Имя-индекса-1 (и -2) может быть установлено с помощью одной из четырех возможностей. Относительный адрес для записи в имя-индекса-1 будет вычислен в соответствии с порядковым но-

мером, заданным непосредственно с помощью целого или идентификатора-3 или косвенно с помощью имени-индекса-3. Если имя-индекса-3 связано с той же самой таблицей, что и имя-индекса-1, то никакого преобразования не происходит, а осуществляется простая пересылка значения. Такая же прямая пересылка значения всегда имеет место для индексного-данного-3.

2. Значение индексного-данного-1 (и -2) может быть установлено только с помощью значения имени-индекса-3 или индексного-данного-3. Для задания начального значения индексного-данного нельзя использовать идентификатор-3 или целое, так как вычисления адреса не производится, а выполняется прямая пересылка значения.

3. Идентификатор-1 (и -2) может быть установлен только с помощью имени-индекса-3. В этом случае относительный адрес из имени-индекса-3 преобразуется в соответствующий порядковый номер и запоминается в идентификаторе-1 как целое.

Таким образом следующие примеры неверны ¹⁾:

SET INDEX-DATA-ITEM-A TO 1. (НЕВЕРНО!)

SET INDEX-DATA-ITEM-A TO ANY-DATA-ITEM. (НЕВЕРНО!)

SET ANY-DATA-ITEM TO 1. (НЕВЕРНО!)

SET ANY-DATA-ITEM TO INDEX-DATA-ITEM-A. (НЕВЕРНО!)

Второй формат оператора SET намного проще. Он используется только для изменения значения имени-индекса (для его увеличения или уменьшения).

Формат 2:

$$\text{SET имя-индекса-1 [имя-индекса-2] ... } \left\{ \begin{array}{l} \text{UP BY} \\ \text{DOWN BY} \end{array} \right\} \left\{ \begin{array}{l} \text{идентификатор} \\ \text{целое} \end{array} \right\}$$

Примеры:

SET INDEX-A UP BY 1.

SET INDEX-A INDEX-B DOWN BY SOME-VALUE.

Для увеличения или уменьшения имени-индекса могут использоваться только числовые величины, причем до изменения имени-индекса осуществляется преобразование порядкового номера в относительный адрес.

Существует еще одно место, где необходимо позаботиться о различении относительных адресов и числовых значений. Это проверка условия в операторе IF, где сравниваются имена-индексов,

¹⁾ INDEX-DATA-ITEM-A означает ИНДЕКСНОЕ-ДАННОЕ-A и служит именем индексного-данного. ANY-DATA-ITEM означает ЛЮБОЕ-ДАННОЕ и служит идентификатором. — *Прим. перев.*

индексные-данные и обычные данные. Все сравнения делаются для порядковых номеров, и, за исключением случая индексных-данных, выполняется соответствующее преобразование. Однако для индексных-данных не делается преобразования относительного адреса, и всякий раз, когда в условии присутствует индексное-данные, выполняется сравнение фактических значений¹⁾. Таким образом, сравнение индексного-данного с литералом или обычным данным невозможно.

Фразу OCCURS с добавлением INDEXED BY совместно с оператором SET можно использовать для написания более коротких процедур, чем без использования этих средств. Ниже приведен пример, в котором в таблице отыскивается имя служащего с наибольшим годовым окладом:

01 RECORD-Q.

05 EMPLOYEE-ARRAY OCCURS 350 TIMES
INDEXED BY INDEX-A INDEX-B.

10 ANNUAL-SALARY PICTURE IS 9(6)V99.

10 NAME-OF-EMPLOYEE PICTURE IS A (20).

·
·
·

START-PARAGRAPH.

READ INPUT-RECORD-FILE RECORD INTO RECORD-Q
AT END GO TO ANOTHER-PARAGRAPH.
SET INDEX-A TO 1.

STEP-ONE.

SET INDEX-B TO INDEX-A.

* ЗАМЕЬТЕ — INDEX-B СОДЕРЖИТ АДРЕС
* НАИБОЛЬШЕГО ГОДОВОГО ОКЛАДА
* (ANNUAL-SALARY), НАЙДЕННОГО К ДАННОМУ
* МОМЕНТУ В ПРОЦЕССЕ ПОИСКА;
* В НАЧАЛЕ ЭТО АДРЕС САМОГО ПЕРВОГО ДАННОГО,
* ЗАТЕМ ОН МЕНЯЕТСЯ КАЖДЫЙ РАЗ, КОГДА
* ВСТРЕЧАЕТСЯ ДАННОЕ ANNUAL-SALARY С
* БОЛЬШИМ ЗНАЧЕНИЕМ, НА АДРЕС ЭТОГО
* ПОСЛЕДНЕГО.

¹⁾ Относительный адрес, являющийся значением индексного-данного, рассматривается как целое число. — Прим. перев.

STEP-TWO.

SET INDEX-A UP BY 1.

IF INDEX-A IS GREATER THAN 350

GO TO END-OF-TABLE-SCAN.

IF ANNUAL-SALARY (INDEX-A) IS GREATER THAN
ANNUAL-SALARY (INDEX-B)

GO TO STEP-ONE ELSE GO TO STEP-TWO.

END-OF-TABLE-SCAN.

DISPLAY NAME-OF-EMPLOYEE (INDEX-B).

.
.
.

В примерах, приведенных до сих пор, имена-индексов записывались точно таким же образом, что и индексы, для которых внутри круглых скобок не допускаются арифметические выражения. Для имен-индексов это ограничение ослаблено и допускается «относительная индексация», т. е. к имени-индекса можно добавить или из него можно вычесть целый числовой литерал. Таким образом, допустимо следующее обращение:

TABLE-A (INDEX-A, INDEX-B + 5, INDEX-C - 3)

Необходимо следить за тем, чтобы во время выполнения такое сложение или вычитание не привело к адресам, выходящим за пределы таблицы. При использовании относительной индексации время выполнения увеличивается за счет преобразования литерала в относительный адрес памяти, и, хотя это преобразование требует меньше времени, чем при использовании индексов, оно уменьшает преимущества использования имен-индексов.

Упражнения

1. Текст на английском языке пробит на перфокартах во всех восьмидесяти колонках. Для простоты предположим, что все знаки препинания опущены. Напишите полную КОБОЛ-программу чтения колоды карт и подготовки списка слов, встречающихся в отперфорированном тексте. Даже если некоторое слово в тексте встречается несколько раз, оно должно присутствовать в этом списке только однажды. Например, для текста

THE CAT IN THE HAT SAT IN THAT HAT

список будет таков:

THE
CAT

IN
HAT
SAT
THAT

Предполагается не более 300 различных слов, и каждое слово будет состоять не более, чем из пятнадцати литер. Не надо никак упорядочивать словарь, просто заносите слова в список в том порядке, в котором они встречаются.

2. В файле сотрудников имя отдельного сотрудника хранится в виде:

01	EMPLOYEE-RECORD.	
05	LAST-NAME	PICTURE IS A(25).
05	FIRST-NAME	PICTURE IS A(10).
05	MIDDLE-INITIAL	PICTURE IS A.
05	FILLER	PICTURE IS X(299).

Например:

SMITHSONIAN ROBERT G

Напишите полную КОБОЛ-программу составления списка имен в более привычном виде, т. е. вначале — имя, затем — инициал с точкой и фамилия:

ROBERT G. SMITHSONIAN

без промежутков, за исключением пробелов по обе стороны от инициала.

8.5. Последовательный поиск в таблице

Программа в конце предыдущего раздела представляла собой пример последовательного поиска в таблице. Такой поиск начинался с фиксированного места (в рассмотренном примере — с начала). При этом элементы таблицы выбирались последовательно до тех пор, пока таблица не была исчерпана. В процессе поиска при выполнении определенных условий (встретился новый большой оклад) предпринимались определенные действия, а именно запоминалось текущее значение имени-индекса. Для того чтобы такую операцию можно было удобно и просто описывать, в КОБОЛе существует оператор SEARCH (ИСКАТЬ). Формат наиболее простого варианта этого оператора приводится ниже.

<u>SEARCH</u>	имя-таблицы	[<u>AT END</u>	повелительные-операторы]
	<u>WHEN</u>	условие	{ повелительные-операторы } <u>NEXT SENTENCE</u>

Оператор **SEARCH** — это единственный оператор КОБОЛа, в котором имя-таблицы может упоминаться без индекса или имени-индекса. Это происходит потому, что глагол **SEARCH** сам управляет увеличением имени-индекса, необходимым для последовательного перебора элементов таблицы. В статье-описания-данного для имени-таблицы в операторе **SEARCH** должно присутствовать добавление **INDEXED BY**. Поиск с помощью оператора **SEARCH** осуществляется только с использованием имени-индекса (и никогда с использованием индекса). В описании имени-таблицы должна присутствовать фраза **OCCURS** или эта фраза должна присутствовать в описании имени-таблицы, в которую вложена рассматриваемая таблица. Для приведенного выше простого формата оператора **SEARCH** имя-индекса, следующее непосредственно за словами **INDEXED BY** в описании таблицы, как раз и изменяется при поиске. Однако это имя-индекса не устанавливается автоматически на начало таблицы. Процедура поиска начинается с текущего значения этого имени-индекса, и вся ответственность за то, что это значение правильное, лежит на программисте. Это сделано в основном затем, чтобы каждый новый поиск в той же таблице не приходилось всегда начинать сначала.

Примером оператора **SEARCH**, работающего с таблицей **EMPLOYEE-ARRAY**, описанной на стр. 332 в предыдущем разделе, является следующее:

```
SET INDEX-A INDEX-B TO 1.
```

```
P-1.
```

```
SEARCH EMPLOYEE-ARRAY
```

```
AT END DISPLAY NAME-OF-EMPLOYEE (INDEX-B)
```

```
WHEN
```

```
ANNUAL-SALARY (INDEX-A) IS GREATER
```

```
THAN ANNUAL-SALARY (INDEX-B)
```

```
SET INDEX-B TO INDEX-A GO TO P-1.
```

Начальная установка **INDEX-A** отделена от оператора **SEARCH**. Все остальные фразы относятся к единственному оператору **SEARCH**. Описание имени-таблицы (в нашем случае **EMPLOYEE-ARRAY**) должно включать фразу **OCCURS** и фразу **INDEXED BY**.

Имя-таблицы в операторе SEARCH может относиться к таблице, вложенной в другую таблицу, но увеличиваться будет только имя-индекса, связанное с указанной таблицей. Поиск всегда будет вестись в направлении увеличения значения имени-индекса.

Первым делом глагол SEARCH проверяет, не превосходит ли значение INDEX-A число 350, являющееся размером таблицы. Если превосходит, то выполняется повелительный-оператор, следующий за словами AT END, если таковой имеется. Если такой повелительный-оператор присутствует, то после его выполнения, а если он отсутствует, то сразу же управление передается следующему предложению программы и продолжается выполнение оставшейся части процедуры. Если же значение имени-индекса находится в допустимых границах таблицы, то проверяется выполнение условия. Данное условие представляет собой любое условие, которое может быть проверено в КОБОЛе: знак, класс (числовой или алфавитный), отношение и т. д. Если условие истинно, то выполняется либо повелительный-оператор, либо NEXT SENTENCE (СЛЕДУЮЩЕЕ ПРЕДЛОЖЕНИЕ). Повелительный-оператор может быть последовательностью операторов, как в примере:

SET INDEX-B TO INDEX-A GO TO P-1.

Если условие истинно и оператор GO TO отсутствует, то поиск прекращается и управление передается следующему предложению. Именно поэтому в рассматриваемом примере вставлен оператор GO TO P-1. Когда поиск завершен нахождением нового большего оклада, его необходимо возобновить. Если условие ложно, то имя-индекса увеличивается на один порядковый номер (преобразованный в адресную величину) и процесс повторяется.

Разрешение использовать несколько необязательных фраз WHEN увеличивает сложность оператора SEARCH. Соответствующие условия будут проверяться в порядке их написания, и, если хотя бы одно из них выполнено, поиск прекращается. Например:

SEARCH TABLE-A

WHEN INDEX-A IS EQUAL TO 10 SET INDEX-B TO 1
WHEN TABLE (INDEX-A) IS NUMERIC GO TO P-1
WHEN TABLE (INDEX-A) IS NEGATIVE STOP RUN.

Заметьте, что этот пример грамматически правилен, но содержит логическую ошибку.

Последнее усложнение оператора SEARCH заключается в разрешении новой альтернативы, задаваемой фразой VARYING (ИЗМЕНЯЯ):


```

SEARCH имя-таблицы
[ VARYING { имя-индекса
             индексное-данное
             идентификатор } ]
[ : AT END повелительные-операторы ]

; WHEN условие { повелительные-операторы }
[ ; WHEN условие { NEXT SENTENCE
                  повелительные-операторы } ] ...
                  NEXT SENTENCE

```

Если имя во фразе VARYING является

1) именем-индекса, связанным с именем-таблицы, будучи в списке имен-индексов, следующих за словами INDEXED BY, то вместо имени-индекса, следующего непосредственно за словами INDEXED BY, оператор SEARCH увеличивает именно его значение;

2) именем-индекса для другой таблицы, индексным-данным или идентификатором (обязательно описанным как целое или упомянутым во фразе USAGE IS INDEX), то используется имя-индекса, стоящее сразу же за словами INDEXED BY, и значение имени, следующего за словом VARYING, увеличивается на величину порядкового номера или соответствующего ему относительного адреса.

Но это индексное-данное или имя-индекса следует установить правильным образом до начала поиска, как в примере:

```
SET INDEX-A TO 1.
```

```
SET INDEX-DATA-ITEM-A TO INDEX-A.
```

```
SEARCH TABLE-A
```

```
VARYING INDEX-DATA-ITEM-A
```

```
WHEN TABLE (INDEX-DATA-ITEM-A) IS NEGATIVE GO
TO P-1.
```

До сих пор число элементов в каждой таблице определялось во фразе OCCURS с помощью части фразы «целое TIMES». В точности столько элементов должно быть в таблице, не больше и не меньше. Однако существует возможность определять таблицы с переменным числом элементов. Это делается путем специального добавления к фразе OCCURS:

Формат 1 фразы OCCURS (старый):

```
OCCURS целое-2 TIMES
```

Формат 2 фразы OCCURS (новый):

OCCURS целое-1 TO целое-2 TIMES
DEPENDING ON имя-данного

Имя-данного может быть уточнено, но не может содержать ни индексов, ни имен-индексов. В тех случаях, когда имя-данного может употребляться в КОБОЛ-программе с индексом или именем-индекса, оно будет упоминаться как идентификатор. Целое-2, присутствующее в обоих форматах, определяет размер области памяти, отводимой под таблицу. Назначение формата 2 состоит в том, чтобы определять число элементов таблицы, доступное в текущий момент. Напомним, что число используемых элементов таблицы определяется текущим значением имени-данного. Значение имени-данного при этом должно принадлежать интервалу от целого-1 (которое может быть нулем) до целого-2 включительно. В качестве примера формата 2 рассмотрим описание

05 ITEM-G PICTURE IS X(4)

OCCURS 1 TO 50 TIMES DEPENDING ON TABLE-SIZE.

Под таблицу ITEM-G будет отведено место для хранения 200 литер, однако в любой текущий момент в процессе выполнения программы значение данного TABLE-SIZE будет определять, сколько элементов может обрабатываться, скажем, оператором SEARCH. Значения элементов таблицы ITEM-G с порядковыми номерами, превосходящими значение TABLE-SIZE, будут недоступны для программы и могут быть уничтожены. Имя-данного может быть включено в ту же статью-описания-данного, что и таблица переменного размера, но должно появиться в этой статье раньше, чем таблица. Например:

01 RECORD-Q.

05 TABLE-SIZE PICTURE IS 999.

05 ITEM-G PICTURE IS X(4)

OCCURS 1 TO 50 TIMES

DEPENDING ON TABLE-SIZE.

Недоступные данные между ITEM-G (TABLE-SIZE) и ITEM-G (50) могут либо включаться, либо не включаться в каждую выдаваемую запись; тот или иной вариант выбирают авторы каждой реализации стандартного КОБОЛа, и рассчитывать на включение этих данных нельзя. Таблица переменной длины не обязательно должна быть последней в статье-описания-данного, но она не может быть подчиненной другой таблице. Следующая статья не верна:

05 A-X OCCURS 3 TIMES.

10 B-X OCCURS 4 TIMES.

15 C-X OCCURS 3 TO 30 TIMES (HEBERHO!)
 DEPENDING ON VALUE-X. (HEBERHO!)

Опять же имена-данных (такие, как TABLE-SIZE или VALUE-X) могут быть уточнены, но не могут быть индексированы. И наконец, повторим, что таблицы переменной длины нельзя использовать для резервирования места, а только для управления процессом внутренней обработки данных. Оператор SEARCH (и оператор SEARCH ALL из следующего раздела) прекратит поиск в таблице при достижении границы, определенной фразой DEPENDING ON. Рассмотрим колоду перфокарт, описанных следующим образом:

01 CARD-RECORD-FORMAT.

05 CARD-SEQUENCE-NUMBER PICTURE IS 9(3).

05 NUMBER-OF-ITEMS PICTURE IS 9(2).

05 VARYING-ITEMS OCCURS 0 TO 15 TIMES

DEPENDING ON NUMBER-OF-ITEMS

INDEXED BY INDEX-CARD.

10 CODE-NUMBER PICTURE IS X(2).

10 NUMBER-BOUGHT PICTURE IS X(3).

Описание требует, чтобы были пробиты только колонки 1—3 и 4—5. Остаток карты может содержать переменное число полей. По мере считывания карт можно осуществлять процедуру поиска единственного вхождения определенного кода, например, таким образом:

SET ITEM-COUNT TO ZERO.

P-1.

READ INPUT-FILE RECORD INTO CARD-RECORD-
 FORMAT AT END GO TO P-6.

SET INDEX-CARD TO 1.

SEARCH VARYING-ITEMS

VARYING ITEM-COUNT

AT END GO TO P-1

WHEN CODE-NUMBER (INDEX-CARD)

IS EQUAL TO "X5" NEXT SENTENCE.

DISPLAY "CODE FOUND ON THE" ITEM-COUNT

"ITEM READ IN".

GO TO P-1.

Заметим, что ITEM-COUNT — это идентификатор, для которого требуется статья-описания-данного, не показанная в этом примере.

Упражнения

1. Предположим, что на ленте существует файл, содержащий 150 000 записей длиной в пятьдесят литер, хранящих информацию об автомобильных авариях, собранную за период в двадцать лет. В одну физическую запись помещается сто логических записей, имеющих следующий формат:

01 DATA-RECORD.

05 AUTOMOBILE-NAME	PICTURE IS X(25).
05 COST-OF-ACCIDENT	PICTURE IS 9(6)V99.
05 YEAR-OF-ACCIDENT	PICTURE IS 9(4).
05 FILLER	PICTURE IS X(13).

Записи в файле расположены в случайном порядке, и при решении рассматриваемой задачи упорядочивать их не нужно. Напишите полную КОБОЛ-программу для чтения этого файла из 150 000 записей и для вычисления и печати средней стоимости аварий (COST-OF-ACCIDENT) за каждый год (YEAR-OF-ACCIDENT) из десяти последних для каждого упомянутого автомобиля. Известно, что число различных имен автомобилей (AUTOMOBILE-NAME) не превосходит 200, но фактически в файле может встретиться намного меньше. Записи об авариях, происшедших более десяти лет назад, игнорируются.

2. Завершите секцию рабочей-памяти и раздел процедур для следующего примера. Предположим, что имеется наряд на работу, который может быть выполнен сотрудником из файла сотрудников. Задача состоит в следующем. Заказ считывается из файла AVAILABLE-JOB-FILE. Он содержит восьмидесятизначный номер профессии (JOB-SKILL-NUMBER). Необходимо просмотреть файл сотрудников и найти, есть ли у кого-либо подходящий номер профессии (OLD-JOB-SKILL). Если такого номера нет, то распечатайте соответствующее сообщение. Если в точности у одного сотрудника оказался подходящий номер, то распечатайте его идентификационный номер (OLD-ID-NUMBER) и имя (OLD-NAME). Если такой номер оказался у нескольких сотрудников, то выберите одного с наивысшей квалификацией (OLD-MERIT-RATING).

Файл сотрудников упорядочен по возрастанию номеров профессий. Однако в пределах группы записей с одним и тем же номером профессии записи расположены случайным образом, т. е. не упорядочены по квалификации. Записей с одним и тем же номером профессии всегда будет не более ста.

IDENTIFICATION DIVISION.

PROGRAM-ID. EXAM.

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER. COMPUTER-NAME.

OBJECT-COMPUTER. COMPUTER-NAME.

INPUT-OUTPUT SECTION.

SELECT AVAILABLE-JOB-FILE ASSIGN TO READER.

SELECT OLD-PERSONNEL-FILE ASSIGN TO TAPE.

SELECT PRINT-FILE ASSIGN TO PRINTER.

DATA DIVISION.

FILE SECTION.

FD AVAILABLE-JOB-FILE

LABEL RECORDS ARE STANDARD.

01 AVAILABLE-JOB-RECORD.

05 JOB-SKILL-NUMBER PICTURE IS 9(8).

05 FILLER PICTURE IS X(72).

FD OLD-PERSONNEL-FILE

LABEL RECORDS ARE STANDARD.

01 OLD-PERSONNEL-RECORD.

05 OLD-ID-NUMBER PICTURE IS 9(10).

05 OLD-NAME PICTURE IS X(25).

05 FILLER PICTURE IS X(6).

05 OLD-JOB-SKILL PICTURE IS 9(8).

05 FILLER PICTURE IS X(20).

05 OLD-MERIT-RATING PICTURE IS 9(3).

05 FILLER PICTURE IS X(10).

FD PRINT-FILE

LABEL RECORDS ARE STANDARD.

01 PRINT-LINE.

05 PRINT-JOB-SKILL PICTURE IS 9(8).

05 FILLER PICTURE IS X(3).

05 PRINT-ID-NUMBER PICTURE IS 9(10).

05 FILLER
05 PRINT-NAME
05 FILLER

PICTURE IS X(3).
PICTURE IS X(25).
PICTURE IS X(3).

WORKING-STORAGE SECTION.

8.6. Непоследовательный поиск в таблице

При последовательном поиске в списке из ста случайных чисел в среднем придется сделать примерно пятьдесят сравнений прежде, чем будет найдено требуемое число. Процедура поиска могла бы быть сделана намного эффективнее, если бы список чисел был упорядочен по их возрастанию. Сравнение искомой величины с числом, находящимся в середине списка, сразу же исключило бы половину чисел из дальнейшего поиска. Если бы искомое число было меньше числа, расположенного в середине списка, то оно должно было бы находиться в первой его половине. Повторное использование такого деления пополам уменьшило бы число возможностей до 25, 12, 6, 3 и, наконец, 1. После чего осталось бы сделать последнее сравнение. С помощью такого специального поиска нужное число было бы найдено не более чем за семь попыток вместо пятидесяти.

Только что описанная процедура называется двоичным поиском. Он может применяться только при условии, что список упорядочен и особенно эффективен для больших таблиц. Например, для таблицы из 5000 элементов одно сравнение исключает 2500 возможностей. Всего только пятнадцати сравнений достаточно для отыскания определенного элемента в таблице из 32 000 элементов. Двоичный поиск — это всего лишь один из возможных специальных методов поиска. Он широко используется по причине относительной простоты его программирования. В языке КОБОЛ имеется средство для реализации этого специального поиска с помощью одного оператора. Это оператор SEARCH ALL (ИСКАТЬ ОСОБО), имеющий формат:

SEARCH ALL имя-таблицы

[AT END побелительные-операторы]

WHEN специальное-условие { побелительные-операторы }
NEXT SENTENCE

По стандартам КОБОЛа не требуется, чтобы для поиска специального-условия был использован именно двоичный поиск; выбор метода поиска остается за авторами компилятора. На самом деле программисту и не нужно знать, какой метод поиска используется,

но во всех современных компиляторах, вероятно, используется двоичный поиск и это объясняет ограничения, налагаемые на таблицы.

Для применения оператора SEARCH ALL требуется, чтобы элементы таблицы были упорядочены в соответствии с одним или более данными, называемыми ключами. Например, эти элементы могут быть упорядочены по двум ключевым данным:

Ключ 1	Ключ 2	Остаток элемента
100	008	...
100	009	...
105	040	...
105	048	...
105	050	...
108	004	...
110	006	...
110	018	...
110	029	...

Элементы упорядочены по возрастанию ключа 1, а для одинаковых значений ключа 1 — по возрастанию ключа 2. Будем говорить, что ключ 1 старше ключа 2. Элементы должны быть упорядочены до применения оператора SEARCH ALL. Это обычно делается путем чтения таблицы в память уже в нужной последовательности. Соответствующий порядок определяется в программе с помощью еще одной альтернативы фразы OCCURS. Дополнительная фраза такова:

$$\left\{ \begin{array}{l} \text{ASCENDING} \\ \text{DESCENDING} \end{array} \right\} \text{ KEY IS имя-донного-1} [\text{имя-донного-2}] \dots$$

Пример описания с новой фразой OCCURS:

```
05 ITEM-X OCCURS 9 TIMES
  INDEXED BY INDEX-A
  ASCENDING KEY IS NAME-X NAME-Y.
10 NAME-X          PICTURE IS 9(3).
10 NAME-Y          PICTURE IS 9(3).
10 REMAINDER-G    PICTURE IS X(15).
```

Для такой таблицы может быть написан следующий оператор SEARCH ALL:

SEARCH ALL ITEM-X

AT END DISPLAY "KEY VALUE NOT IN ARRAY"

WHEN NAME-X (INDEX-A) IS EQUAL TO 108

MOVE REMAINDER-G (INDEX-A) TO PRINT-LINE.

Оператор SEARCH ALL выглядит почти так же, как и оператор SEARCH предыдущего раздела, но между ними существуют важные различия. Прежде всего различен метод поиска. При последовательном поиске будет проверяться каждый элемент последовательности до тех пор, пока не будет обнаружено значение 108 в таблице NAME-X. При непоследовательном поиске в операторе SEARCH ALL будет проверен пятый элемент и будет обнаружено, что 108 больше 105. Следующим будет проверен седьмой элемент (так как в этом случае нет срединного элемента), а затем — шестой элемент, сравнение с которым приведет к выполнению условия. После этого будет выполнен повелительный-оператор, следующий за WHEN. Другие различия заключаются в том, что здесь нет фразы VARYING и допускается только одна фраза WHEN. Но наиболее существенное для пользователя различие состоит в том, что специальное-условие оператора SEARCH ALL не может быть произвольным условием, как в операторе SEARCH. В операторе SEARCH ALL может быть сделана лишь совершенно определенная проверка, а именно: может быть сделано сравнение на совпадение одного или более имен-ключей с константами и ничего более. Основное сравнение таково:

$$\text{имя-ключа} \left\{ \begin{array}{c} \text{IS EQUAL TO} \\ = \end{array} \right\} \left\{ \begin{array}{c} \text{целое} \\ \text{идентификатор} \end{array} \right\}$$

Порядок, показанный в этом определении специального-условия, важен; имя-ключа должно быть слева и только слева. Таким образом, условие

NAME-X (INDEX-A) IS EQUAL TO 43

записано верно, а условие

43 IS EQUAL TO NAME-X (INDEX-A) (НЕВЕРНО!)

не допускается синтаксисом языка. Может быть проверено только равенство. При непоследовательном поиске ищется в точности равное значение. Для формирования специального-условия основные сравнения можно комбинировать, объединяя их с помощью союза AND (И), например:

KEY-1 (INDEX-A) = 500 AND KEY-2 (INDEX-A) = 403

Для объединения может быть использован только союз AND; условия OR (ИЛИ) не допускаются. Если в специальном-условии

упоминается какое-либо имя-ключа, то в нем должны быть упомянуты все имена-ключей, старшие по отношению к данному. Старшие имена-ключей во фразе OCCURS перечисляются первыми. В приведенном выше примере NAME-X старше, чем NAME-Y. Старшинство понимается в смысле упорядочивания. Значения ключа NAME-Y упорядочены для каждого из значений ключа NAME-X.

Значения имен-ключей в элементах таблицы не обязательно должны быть целыми числами, так как для определения их вида можно использовать любую фразу PICTURE. Но элементы должны быть упорядочены по этим значениям, и если эти значения отличаются от числовых или алфавитных, то может возникнуть проблема совместности, так как на различных машинах специальные символы или алфавитно-цифровые величины могут быть упорядочены по-разному. В примере приведены два имени-ключа, оба в возрастающем порядке. На самом деле может быть любое число имен-ключей и порядок каждого, независимо от порядка других, может быть возрастающим или убывающим. Имена-ключей могут располагаться в любом месте записи, а не только в начале. Существует только одно ограничение на расположение имен-ключей: они не могут следовать за переменной частью элемента (т. е. за фразой DEPENDING ON). В этом смысле они подобны имени-данного во фразе DEPENDING ON. Имя-ключа должно располагаться в фиксированной части элемента. Кроме того, имена-ключей должны быть описаны в описании имени-таблицы, упомянутой в операторе SEARCH ALL. Первое имя-ключа может быть самым именем-таблицы или любым подчиненным именем-данного, а все последующие имена-ключей должны быть подчинены имени-таблицы и каждое из них должно быть старшим по отношению к следующему имени-ключа.

При непоследовательном поиске изменяется значение первого имени-индекса, следующего за словами INDEXED BY во фразе OCCURS рассматриваемой таблицы. Возможность изменить какое-либо другое имя-индекса с помощью дополнения VARYING отсутствует. После успешного завершения поиска (специальное-условие выполнено) значение этого первого имени-индекса устанавливается равным относительному адресу элемента, для которого выполнялось условие, и управление передается следующему предложению (вариант NEXT SENTENCE) или исполняется повелительный-оператор. Если этот оператор не GO TO, то управление также передается следующему предложению после исполнения повелительного-оператора. В случае когда при поиске нужный элемент не найден, значение имени-индекса не определено и управление передается оператору, следующему за словами AT END (или непосредственно следующему предложению, если слова AT END отсутствуют).

Если имя-ключа имеет несколько одинаковых значений (как в нашем примере, имя-ключа NAME-X), удовлетворяющих специальному-условию, то значение имени-индекса будет правильным, но нельзя сказать, какому из элементов, удовлетворяющих специальному-условию, оно соответствует.

Упражнения

1. Для использования оператора SEARCH ALL необходимо, чтобы элементы таблицы были упорядочены либо в возрастающей, либо в убывающей последовательности. Такая последовательность не создается автоматически, так что необходимо написать процедуру упорядочивания элементов. Предположим, что в программу ранее была загружена таблица с некоторыми значениями элементов, имеющая описание:

01 TABLE-OF-ELEMENTS.

05 ELEMENT-X OCCURS 500 TIMES
INDEXED BY INDEX-A
ASCENDING KEY IS KEY-A.

10 KEY-A PICTURE IS 9(3).

10 REMAINDER-X PICTURE IS X(50).

Эти значения не обязательно были получены в каком-либо определенном порядке. Напишите процедуру переупорядочивания 500 значений элементов таблицы ELEMENT-X, так чтобы значения ключа KEY-A были расположены в возрастающей последовательности.

2. Предположим, что запись TABLE-OF-ELEMENTS предыдущего упражнения упорядочена по возрастанию ключей. Предположим, что имеется файл записей следующего формата:

01 INPUT-RECORD.

05 TEST-VALUE PICTURE IS 9(3).

05 FILLER PICTURE IS X(60).

05 CHECK-X PICTURE IS X(50).

Напишите процедурный сегмент для чтения, записи и поиска в таблице элемента со значением ключа KEY-A, равным значению данного TEST-VALUE. Если часть REMAINDER-X элемента таблицы равна данному CHECK-X, то распечатайте запись; если нет, то перейдите к следующей записи. Повторяйте этот процесс до тех пор, пока файл не будет исчерпан.

3. Напишите полную КОБОЛ-программу присвоения начальных значений элементам следующей таблицы:

01 TABLE-X.

05 ENTRY-X OCCURS 120 TIMES
INDEXED BY INDEX-A
DESCENDING KEY IS YEAR-MONTH.
10 YEAR-MONTH.
15 YEAR-X PICTURE IS 9(2).
15 MONTH-X PICTURE IS 9(2).
10 CHECK-DIGIT PICTURE IS 9.

таким образом, чтобы все значения CHECK-DIGIT были равны нулю, а значения YEAR-MONTH (ГОД-МЕСЯЦ) начинались с текущего месяца и следовали в обратном порядке на десять лет назад. После установки начального состояния таблицы прочитайте файл записей, содержащих даты. Для каждой записи проверьте, имеется ли в таблице ее дата (конечно, только год и месяц); если да, то установите значение соответствующего данного CHECK-DIGIT равным единице. После исчерпания файла просмотрите таблицу и распечатайте все даты, для которых CHECK-DIGIT осталось равно нулю.

Глава 9. Средства эффективного программирования на КОБОЛе

9.1. Фраза USAGE

Фраза USAGE (ДЛЯ) используется в статье-описания-данного для задания способа кодировки в памяти значения данного. Фраза USAGE может быть написана как на элементарном, так и на групповом уровне. На групповом уровне эта фраза относится к каждому подчиненному элементарному данному. Не должно быть противоречия между групповой фразой USAGE и любой подчиненной фразой USAGE, написанной внутри соответствующей группы.

Формат фразы USAGE таков:

$$[\text{USAGE IS}] \left\{ \begin{array}{l} \text{COMPUTATIONAL} \\ \text{COMP} \\ \text{DISPLAY} \\ \text{INDEX} \end{array} \right\}$$

Слово COMP (ВЫЧ) — это сокращение слова COMPUTATIONAL (ДЛЯ ВЫЧИСЛЕНИЙ). Оба эти слова означают одну и ту же возможность. Поэтому существует только три способа представления данных. Если фраза USAGE опущена, то предполагается, что данное представлено в виде DISPLAY (ВЫВОД). Таким образом, все данные в примерах, приводимых до сих пор, были представлены в виде DISPLAY, за исключением данных, описанных как INDEX.

Целью этих трех представлений является использование для данных специальной кодировки в соответствии с характером их использования. Элементы данных, описанные как INDEX, являются индексными-данными, а их значения — это адреса памяти, имеющие длину и кодировку, определяемую авторами компилятора. Программист не должен знать, где в памяти расположены эти данные и сколько места требуется для закодированного представления адреса памяти. Элементарные индексные-данные нельзя использовать в операторе MOVE, но они могут быть частью группы, к которой применяется этот оператор. В этом случае осуществляется пересылка в виде литер без всякого преобразования.

Данные, описанные как DISPLAY либо явно, либо по умолчанию, представлены с помощью некоторого битового кода, длину (количество бит в одной литере) которого могут выбрать авторы компилятора по своему усмотрению. В этом случае каждая цифро-

вая, буквенная или специальная литера занимает в точности одну литерную позицию. Единственным исключением является знак числа, который заключается в одну из позиций цифровых литер (чаще всего последнюю). Заметим, что фраза SIGN (ЗНАК) может использоваться только для данных вида DISPLAY. Максимальная длина числового данного — восемнадцать десятичных цифр; буквенных или буквенно-цифровых данных — 120 литер. В этом случае данные распечатываются в том же виде, в котором они хранятся в памяти. Все данные, хранящиеся в машине, должны быть представлены с помощью двоичных цифр (0 или 1); числовое данное вида DISPLAY должно быть закодировано четырехбитовым кодом, например, следующим образом:

Значение данного:	1	2	5
Представление:	0001	0010	0101

Элементы данных, описанные как COMPUTATIONAL, используются в арифметических операциях и должны быть числовыми. Строка шаблона должна быть составлена только из символов 9 с необязательным S для знака, V для предполагаемой десятичной точки или P для десятичного порядка. Описание COMPUTATIONAL позволяет авторам компилятора хранить значение данного в любом виде, который наиболее эффективен для конкретной машины. В большинстве случаев это означает преобразование числового значения в систему счисления с основанием, отличным от десяти, и, возможно, использование специального обозначения для работы с десятичным порядком. Использование способа COMPUTATIONAL уменьшает объем необходимой памяти и ускоряет выполнение арифметических операций приблизительно в четыре раза. Программист не сможет сказать, сколько потребуется памяти (количество необходимой памяти часто будет меняться в зависимости от значения данного). Данные вида COMPUTATIONAL не следует использовать в записях, которые будут записываться в файлы. Однако данные этого вида можно выдавать с помощью оператора DISPLAY, так как размер таких записей не фиксируется. Пример описания данного вида COMPUTATIONAL:

```
77 NUMBER-X VALUE IS +5.25 PICTURE IS S99V99
   USAGE IS COMPUTATIONAL.
```

Повторим, что программист не сможет визуально узнать это значение. Например, значение 0525 скорее всего будет храниться как-нибудь так:

0000010100100101

Рекомендуется читать данные из внешнего файла в запись, специфицированную как буквенно-цифровая вида DISPLAY, а

затем для обработки перенести эти данные в запись, состоящую из данных вида COMPUTATIONAL. Оператор MOVE осуществит преобразование вида DISPLAY к виду COMPUTATIONAL. Вычисления можно выполнять над величинами, помещенными в рабочую-память, получая, таким образом, преимущество от представления данных в виде, удобном для вычислений. После этого результаты вычислений можно вновь поместить в выходную запись вида DISPLAY с еще одним автоматическим преобразованием вида и переписать в файл уже в литерной форме (USAGE IS DISPLAY). Например:

FD INPUT-FILE

LABEL RECORD IS STANDARD.

01 RECORD-X.

05 ITEM-ONE PICTURE IS X(5) USAGE IS DISPLAY.

05 ITEM-TWO PICTURE IS X(6) USAGE IS DISPLAY.

05 ITEM-THREE PICTURE IS X(4) USAGE IS DISPLAY.

FD OUTPUT-FILE

LABEL RECORD IS STANDARD.

01 RECORD-Y.

05 ITEM-ONE PICTURE IS 9(5) USAGE IS DISPLAY.

05 ITEM-TWO PICTURE IS 9(6) USAGE IS DISPLAY.

05 ITEM-THREE PICTURE IS 9(4) USAGE IS DISPLAY.

WORKING-STORAGE SECTION.

01 COMPUTATIONAL-RECORD.

05 ITEM-ONE PICTURE IS S9(5) USAGE IS
COMPUTATIONAL.

05 ITEM-TWO PICTURE IS S9(6) USAGE IS
COMPUTATIONAL.

05 ITEM-THREE PICTURE IS 9(4) USAGE IS
COMPUTATIONAL.

•
•
•

PROCEDURE DIVISION.

READ INPUT-FILE RECORD AT END GO TO P-6.
MOVE CORRESPONDING RECORD-X TO

COMPUTATIONAL-RECORD.

SUBTRACT ITEM-ONE IN COMPUTATIONAL-RECORD
FROM ITEM-TWO IN COMPUTATIONAL-RECORD.

MOVE CORRESPONDING COMPUTATIONAL-RECORD TO
RECORD-Y.

WRITE RECORD-Y.

.
.
.

Таким образом, считается, что во внешнем файле лучше не пытаться хранить информацию в виде COMPUTATIONAL. Программисты, которые пытаются использовать свое знание конкретного способа кодирования, примененного авторами компилятора, с целью экономии места в файлах, записывая в них данные в виде COMPUTATIONAL, постоянно испытывают затруднения, связанные с тем, что сформированные файлы не могут быть использованы на других вычислительных машинах или других операционных системах. К тому же другие программисты, которые должны использовать эти файлы в своих программах, сталкиваются при этом с существенными трудностями.

Таким образом, стандартное требование состоит в том, чтобы все входные и выходные записи были специфицированы фразой USAGE IS DISPLAY либо явно, либо по умолчанию.

9.2. Синхронизация

Внутренняя память машины конструктивно разбивается на конечные группы битов, называемые словами, которые участвуют в перемещениях и других машинных командах как единое целое. Слова задают естественные границы, которые необходимо учитывать при размещении информации в памяти ЭВМ, так как если данное можно целиком поместить в одно слово, то передача данных в ЭВМ упрощается. Размер слов и битовое представление информации существенно различаются на различных машинах. Рассмотрим задачу помещения на сумматор для выполнения арифметической операции числа 392604. Если бы это число размещалось в памяти так:

слово # 1	000000
слово # 2	392604

то было бы достаточно единственной пересылки слова #2 на сумматор. При этом предполагается, что в каждом слове памяти хранится шесть стандартных литер данных. Однако статья-описания-данного может быть такова:

```
05 FIRST-DATA    PICTURE IS 9(3)    VALUE IS ZERO.
05 SECOND-DATA   PICTURE IS 9(6)    VALUE IS 392604.
```

при этом распределение памяти будет следующим:

```
слово # 1    000392
слово # 2    604000
```

За тремя нулями данного FIRST-DATA сразу же будет следовать значение 392604 данного SECOND-DATA, пересекая естественную границу между словом #1 и словом #2. Теперь, для того чтобы поместить число 392604 на сумматор как единую числовую величину, потребуются не только два перемещения, но и операции перестановки и сдвига. Для каждой машины возможны ситуации, при которых значения данных могут пересекать границы элементов памяти. В статью описания элементарного данного может быть добавлена фраза, которая приведет к выравниванию значений данных относительно стандартного разбиения памяти на слова. Это фраза синхронизации, имеющая такой формат:

$$\left\{ \begin{array}{c} \text{SYNCHRONIZED} \\ \text{SYNC} \end{array} \right\} \left[\begin{array}{c} \text{LEFT} \\ \text{RIGHT} \end{array} \right]$$

Заметьте, что слова LEFT (ВЛЕВО) и RIGHT (ВПРАВО) не обязательны и могут быть оба опущены. В этом случае значение данного располагается в слове памяти так, как определено авторами компилятора для достижения наибольшей эффективности. Использование слов LEFT или RIGHT означает, что данное сдвинуто к левой или правой границе стандартного слова. Лучше всего использовать фразу SYNCHRONIZED (ВЫДЕЛЕНО) без слов LEFT или RIGHT, так как программист не может определить наилучший способ расположения данных для каждой машины.

Если бы в приведенном ранее примере второе данное имело описание:

```
05 SECOND-DATA PICTURE IS 9(6) VALUE IS 392604
    SYNCHRONIZED.
```

то число 392604 было бы целиком размещено в слове #2, а между тремя цифрами данного FIRST-DATA и шестью цифрами данного SECOND-DATA образовался бы зазор. Размер этого зазора не известен и не может быть предсказан безотносительно к конкретной вычислительной машине подобно тому, как и в случае фразы USAGE IS COMPUTATIONAL не может быть предсказан объем памяти.

Поскольку сдвиг при синхронизации не известен, эту фразу не следует применять в записи, которая будет записываться в файлы. При выполнении операции MOVE данные будут не только редактироваться и преобразовываться, но и будут размещаться соответствующим образом. На сам же размер данной синхронизации не влияет, так как этот размер по-прежнему определяется фразой PICTURE. Но синхронизация влияет на суммарный объем используемой памяти, точнее на то, как элементарные данные будут разделяться зазорами. Возможно, что программисту придется решать, компенсируется ли потеря памяти из-за зазоров выигрышем во времени, полученным за счет синхронизации. Например, в синхронизированной таблице зазоры могут быть между всеми элементарными данными, что приведет к суммарному большому расходу памяти. Рекомендуется использовать синхронизацию тогда, когда данные имеют вид COMPUTATIONAL. Например:

05 GROUP-X OCCURS 20 TIMES.

10 VALUE-A PICTURE IS S9(6)V99
 USAGE IS COMPUTATIONAL
 SYNCHRONIZED.

10 VALUE-B PICTURE IS X(12).

Синхронизация может применяться к любым элементарным данным как числовым, так и буквенно-цифровым, но обычно ее применяют к числовым данным, используемым для вычислений. Кроме того, лучше не переопределять синхронизированное данное, так как для фразы REDEFINES (ПЕРЕОПРЕДЕЛЯЕТ) требуется точное соответствие размещения в памяти новых данных со старыми. При синхронизации такое соответствие может нарушаться. Даже если его можно было бы установить в отдельных случаях, величина зазоров могла оказаться другой при компиляции исходной программы на машине с другим размером слова. Следуя предписаниям предыдущего раздела, нужно употреблять фразу SYNCHRONIZED вместе с фразой USAGE IS COMPUTATIONAL в секции WORKING-STORAGE SECTION. Обычно бесполезно ее использовать совместно с фразой USAGE IS DISPLAY. Использовать синхронизацию вместе с фразой USAGE IS INDEX запрещено.

Необходимо помнить, что групповое перемещение отличается от элементарного перемещения. При групповом перемещении группа данных рассматривается как одно буквенно-цифровое данное. При этом не происходит никакого преобразования и никакого перераспределения элементарных данных в группе. Это не относится к оператору MOVE CORRESPONDING, который определяет серию элементарных перемещений. Попытка переместить групповое данное так, как показано ниже в примере, либо вызовет выход из компилятора по ошибке, либо приведет к неправильным результатам.

Так как диагностические возможности различных компиляторов существенно отличаются, то возможно, что такая ошибка останется необнаруженной.

•
•
•

FD INPUT-FILE

LABEL RECORD IS STANDARD.

01 INPUT-RECORD PICTURE IS X(100).

•
•
•

WORKING-STORAGE SECTION.

01 INTERNAL-RECORD.

05 GROUP-A PICTURE IS X(50).

05 GROUP-B PICTURE IS 9(10) SYNCHRONIZED

USAGE IS COMPUTATIONAL.

05 GROUP-C PICTURE IS X(40).

•
•
•

READ INPUT-FILE RECORD INTO INTERNAL-RECORD
(НЕВЕРНО!)

AT END STOP RUN.

Оператор READ...INTO включает групповое перемещение, и скорее всего объем памяти, отведенной для записи INPUT-RECORD, будет отличаться от объема памяти, нужного для записи INTERNAL-RECORD.

9.3. Условия VALUE

Зарезервированное слово КОБОЛа VALUE может быть использовано в нескольких различных местах программы и иметь различный смысл в зависимости от контекста. Оно уже было описано во фразе VALUE OF (ЗНАЧЕНИЕ) статьи-описания-файла для спецификации значения данного в качестве системной метки или метки пользователя. Второе место — это статьи-описания-данных в сек-

ции WORKING-STORAGE, где определяются начальные значения данных (т. е. значения, которые будут иметь данные сразу после начальной загрузки программы в память). Слово VALUE может быть также использовано для определения значений, связанных со специальными данными, именуемыми статьями-имени-условия. Статья-имени-условия начинается с номера-уровня 88, за которым следует имя-условия и связанное с ним значение, набор значений или диапазон значений. В дальнейшем имя-условия можно использовать как сокращение для проверки того, что действительно встретилось какое-либо из описанных значений. Эти статьи с номером уровня 88 используются для определения данного, называемого условной-переменной, и должны следовать сразу же за статьей, описывающей условную-переменную. Полное описание условной переменной имеет вид:

номер-уровня	условная-переменная	описательные-фразы
88	имя-условия	
	$\left\{ \begin{array}{l} \text{VALUE IS} \\ \text{VALUES ARE} \end{array} \right\}$	литерал-1 [THRU литерал-2]
		[, литерал-3 [THRU литерал-4]]...

За условной-переменной может следовать столько статей уровня 88, сколько необходимо задать условий. Например:

77	VARIABLE-A	PICTURE IS S9(4)V99.
88	ZIP-X	VALUE IS ZERO.
88	UNITY-X	VALUE IS 1.
88	SPREAD-X	VALUE IS 2 THRU 5.
88	TEST-X	VALUES ARE .95 .97 .99.
88	SERIES-X	VALUES ARE 10 THRU 100 500 600 THRU 800.

VARIABLE-A — это условная-переменная, а PICTURE IS S9(4)V99 — ее описательная-фраза. Имена-условий: ZIP-X, UNITY-X, SPREAD-X, TEST-X, SERIES-X. Когда значение VARIABLE-A в результате применения оператора MOVE или в результате других арифметических операций становится равным нулю, то ZIP-X принимает значение *истина*. TEST-X принимает значение «истина» тогда, когда VARIABLE-A становится равным .95, .97 или .99. Во всех остальных случаях значением TEST-X является *ложь*. Назначение имени-условия состоит в обозначении условия при проверке его истинности или ложности в операторе IF. Оно дает возможность с помощью единственного слова задать широкий диапазон проверок. Так, оператор

IF ZIP-X GO TO P-6.

эквивалентен оператору

IF VARIABLE-A IS EQUAL TO ZERO GO TO P-6.

В данном случае достигаемая экономия сомнительна. Но следующий пример демонстрирует преимущество условной переменной. Оператор

IF SERIES-X GO TO P-7.

может быть употреблен вместо оператора

```
IF    VARIABLE-A IS EQUAL TO 10
OR    (VARIABLE-A IS GREATER THAN 10 AND
      VARIABLE-A IS LESS THAN 100)
OR    VARIABLE-A IS EQUAL TO 100
OR    VARIABLE-A IS EQUAL TO 500
OR    VARIABLE-A IS EQUAL TO 600
OR    (VARIABLE-A IS GREATER THAN 600 AND
      VARIABLE-A IS LESS THAN 800)
OR    VARIABLE-A IS EQUAL TO 800
      GO TO P-7.
```

Статья-имени-условия специального уровня 88 может следовать за любым описанием данного (условной-переменной) с любым номером уровня, за исключением следующих случаев:

- 1) номер-уровня 66 или 88;
- 2) индексное-данные;
- 3) любое данное, содержащее фразы JUSTIFIED RIGHT, SYNCHRONIZED или USAGE IS COMPUTATIONAL.

В частности, статья-имени-условия может следовать за элементарным данным в таблице; в этом случае имя-условия должно индексироваться при каждом обращении к нему. Таким образом, если бы статья-описания-данного (или, можно сказать, статья-описания-записи, так как номер-уровня 88 можно использовать в секции файлов) была такова:

01 TABLE-A.

```
05 ENTRY-X    OCCURS 10 TIMES    PICTURE IS 9(4).
      88 SPECIAL-X    VALUE IS 100 THRU 400.
```

то имя-условия SPECIAL-X относилось бы к каждому из 10 элементов таблицы TABLE-A. Значение отдельного элемента ENTRY-X должно было бы проверяться, например, так:

IF SPECIAL-X (SUB-A) GO TO P-8.

Но оператор

IF SPECIAL-X GO TO P-8. (НЕВЕРНО!)

не имеет смысла.

Статья-имени-условия может также использоваться совместно с фразой REDEFINES:

01 RECORD-Z.

05 ITEM-ONE PICTURE IS X(2).

88 CORRECT-ENTRY VALUE IS "AA".

05 ITEM-TWO REDEFINES ITEM-ONE.

10 SINGLE-LETTER-ONE PICTURE IS X.

88 B-FOR-BRAVO VALUE IS "B".

10 SINGLE-LETTER-TWO PICTURE IS X.

88 C-FOR-CHARLIE VALUE IS "C".

В этом случае условие

IF CORRECT-ENTRY

служит для проверки того, что в первых двух позициях записи RECORD-Z стоят буквы AA. Условие

IF B-FOR-BRAVO

служит для проверки присутствия буквы В в первой позиции и игнорирует вторую литеру. Условие

IF C-FOR-CHARLIE

проверяет наличие буквы С во второй позиции.

В приведенных примерах предполагалось, что литералы, используемые после слов VALUE IS, должны быть согласованы с фразой PICTURE и что в подходящих случаях можно использовать стандартные константы (ZERO). При задании диапазона возможных значений необходимо, чтобы первый литерал был меньше второго, например:

VALUE IS 12 THRU 15

Допустимо лишь задание одного значения, набора отдельных значений и диапазона значений. Не разрешается использовать операторы отношения, так что условие

VALUE IS GREATER THAN 200 (НЕВЕРНО!)

запрещено.

На рис. 9.1 приведен пример программы, использующей слова VALUE IS в статье-имени-условия и для задания начального значения данного. В этом примере также показано использование данных вида COMPUTATIONAL и фразы SYNCHRONIZED.

```

IDENTIFICATION DIVISION.
PROGRAM-ID.    PROG930.

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.
SOURCE-COMPUTER.    COMPUTER-NAME.
OBJECT-COMPUTER.    COMPUTER-NAME.

INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT INPUT-FILE
        ASSIGN TO MAGNETIC-TAPE-UNIT
        RESERVE 4 AREAS
        ACCESS MODE IS SEQUENTIAL
        ORGANIZATION IS SEQUENTIAL.
    SELECT OUTPUT-FILE
        ASSIGN TO MAGNETIC-TAPE-UNIT
        RESERVE 1 AREA
        ACCESS MODE IS SEQUENTIAL
        ORGANIZATION IS SEQUENTIAL.

I-O-CONTROL.
    SAME RECORD AREA FOR INPUT-FILE OUTPUT-FILE.

DATA DIVISION.

FILE SECTION.
FD INPUT-FILE
    RECORD CONTAINS 13 CHARACTERS
    BLOCK CONTAINS 1300 CHARACTERS
    LABEL RECORDS ARE STANDARD
    DATA RECORD IS INPUT-RECORD.

01 INPUT-RECORD.
    05 ITEM-1    PICTURE IS 9(3)    USAGE IS DISPLAY.
    88 RECORD-TYPE-ONE  VALUE IS 1 THRU 50.
    05 ITEM-2    PICTURE IS X(10)   USAGE IS DISPLAY.

```

Рис. 9.1. Пример использования фразы COMPUTATIONAL.

Программа PROG930 считывает записи из файла INPUT-FILE и для некоторых записей, идентифицируемых значением данного ITEM-1, порождает новую запись, которая выводится в файл OUTPUT-FILE. Оба файла расположены на катушках магнитной ленты и имеют последовательную организацию. Для файла INPUT-FILE выделено четыре буферных области во внутренней памяти, а для файла OUTPUT-FILE только одна такая область. Выходные записи создаются лишь эпизодически, в то время как прочитана будет каждая входная запись. В параграфе I-O-CONTROL указано SAME RECORD AREA FOR INPUT-FILE и FOR OUTPUT-FILE (ОБЩАЯ ЗОНА ЗАПИСИ ДЛЯ INPUT-FILE и ДЛЯ OUTPUT-

```

FD OUTPUT-FILE
  RECORD CONTAINS 28 CHARACTERS
  BLOCK CONTAINS 280 CHARACTERS
  LABEL RECORDS ARE STANDARD
  DATA RECORD IS OUTPUT-RECORD.

01 OUTPUT-RECORD.
  05 DATA-A    PICTURE IS X(16)    USAGE IS DISPLAY.
  05 DATA-B    PICTURE IS X(12)    USAGE IS DISPLAY.

WORKING-STORAGE SECTION.

01 INTERNAL-RECORD.
  05 LABEL-A    PICTURE IS X(16)    USAGE IS DISPLAY
           VALUE IS "RECORD EXTRACTED"
  05 NAME-A     PICTURE IS 9(10)
           USAGE IS COMPUTATIONAL SYNCHRONIZED.

PROCEDURE DIVISION.

INITIAL-X SECTION.
ONE-X.    OPEN INPUT INPUT-FILE  OUTPUT OUTPUT-FILE.
TWO-X.    READ INPUT-RECORD RECORD AT END GO TO THREE-X.
           IF RECORD-TYPE-ONE MOVE ITEM-2 TO NAME-A
           ADD 500.0 TO NAME-A
           MOVE LABEL-A TO DATA-A
           MOVE NAME-A TO DATA-B
           WRITE OUTPUT-RECORD
           GO TO THREE-X
           ELSE GO TO TWO-X.

THREE-X.  CLOSE INPUT-FILE  OUTPUT-FILE.
           STOP RUN.

```

Рис. 9.1. Пример использования фразы COMPUTATIONAL (продолжение).

FILE), однако для записей такого малого размера это совмещение областей, отводимых под записи, не даст большой экономии и в реальной программе от этого совмещения можно было бы отказаться. Эта статья приведена здесь в качестве примера, а также для того, чтобы подчеркнуть, что две области записей не обязательно должны быть одного и того же размера. Обычно статья SAME используется в том случае, когда буферные области для разных файлов совмещены, но при этом в каждый момент времени может быть открыт только один из файлов, упомянутых во фразе SAME.

В статье-описания-записи для INPUT-RECORD присутствует данное уровня 88 с фразой VALUE IS. Значением данного ITEM-1

в INPUT-RECORD может быть одно из тысячи возможных от 000 до 999. Для любого значения от 001 до 050 включительно при проверке в операторе IF значением имени-условия RECORD-TYPE-ONE будет «истина». Таким образом, второе предложение параграфа TWO-X IN INITIAL-X будет порождать выходную запись только в том случае, когда значение данного ITEM-1 заключено между 001 и 050, включая границы. Оператор MOVE ITEM-2 TO NAME-A правилен, пока все литеры данного ITEM-2 — цифры.

Входные данные преобразуются к виду COMPUTATIONAL с помощью оператора MOVE, и арифметическая операция сложения 500.0 с NAME-A будет выполняться эффективно. Конечно, эта программа не более чем пример, и ради экономии времени только в одной операции сложения на самом деле не стоило бы осуществлять преобразование данных из вида DISPLAY в вид COMPUTATIONAL, а затем обратно. Отметим, что для обратного преобразования требуются элементарные перемещения. Оператор WRITE OUTPUT-RECORD FROM INTERNAL-RECORD здесь не работал бы, так как задавал бы групповое перемещение.

Заменяя предложение STOR RUN в параграфе THREE-X на FOUR-X.

EXIT PROGRAM.

FIVE-X. STOP RUN.

эту программу можно сделать либо самостоятельной, либо вызываемой из некоторой другой программы.

Упражнения

I. Напишите статьи-описания-данных и процедуру для чтения следующей записи:

01 IN-REC.

05 VALUE-A PICTURE IS X(5).

05 VALUE-B PICTURE IS X(5).

05 VALUE-C PICTURE IS X(5).

в которой каждое данное, хотя и описано как буквенно-цифровое, содержит только пять цифр (без пробелов) и представляет собой доллары и центы. При этом 00498 означает четыре доллара и девятью восемь центов. Запомните эти данные в рабочей-памяти в виде, удобном для вычислений, сложите все эти данные и возьмите пятьдесят процентов от суммы в качестве FINAL-ANSWER (ОКОНЧАТЕЛЬНЫЙ-ОТВЕТ). Запомните значение FINAL-ANSWER в виде цифрового отредактированного данного, состоящего из одних пробелов, если значение равно нулю, и содержащего знак доллара и десятичную точку в противном случае. Обратите внимание, что 00498 в виде буквенно-цифрового данного есть целое

число и его нельзя просто поместить в данное с шаблоном 9(3)V9(2) без усечения.

2. Напишите статью-описания-записи и процедуру для проверки числовых оценок студентов (от 000 до 100) и выставления буквенных оценок (LETTER-GRADE) в соответствии с таблицей

A	от 90 до 100
B	от 80 до 89
C	от 70 до 79
D	от 60 до 69
F	в остальных случаях

3. В упражнении из предыдущей главы было введено понятие гибкости при вводе данных, допускающей расположение слова в произвольном месте специфицированного поля. В этом упражнении дано дальнейшее развитие этого понятия. Оно связано с возможностью записывать число с явной десятичной точкой и иметь программное преобразование этой буквенно-цифровой величины в числовую, например:

PICTURE IS X(15)	PICTURE IS 9(7)V9(3)
3.5	0000003500
3.5	0000003500
1234.5	0001234500
300.003	0000300003
400.	0000400000

Приведенные здесь буквенно-цифровые поля состоят из последовательности цифр (которая возможно содержит десятичную точку), заключенной в окружение из пробелов. Напишите процедуру КОБОЛа для преобразования их в числовые нецелые данные с шаблоном 9(7)V9(3).

9.4. Составные условия

Условие описывает ситуацию, оцениваемую во время выполнения, и используется для управления процессом выполнения программы в операторах IF, SEARCH и PERFORM. В качестве примера оператора IF можно привести оператор, рассмотренный в разд. 4.5:

```
IF A-X IS GREATER THAN B-X GO TO P-6
ELSE GO TO P-7.
```

Примером оператора SEARCH мог бы служить оператор, рассмотренный в разд. 8.6 и 8.7:

SEARCH ALL TABLE-G AT END GO TO P-8
 WHEN VAR-W (INDEX-A) IS EQUAL TO 32.5
 MOVE "FINISHED" TO OUTPUT-ITEM.

Оператор PERFORM еще не был описан, это будет сделано позднее в данной главе. Условия были введены в разд. 4.4. К ним относятся:

- 1) условие отношения;
- 2) условие класса;
- 3) условие имя-условия;
- 4) условие знака.

Эти четыре вида условий являются простыми условиями. С помощью круглых скобок и знаков логических операций (NOT, AND и OR) простые условия можно объединять, образуя составные условия в соответствии со следующими правилами:

- 5) (условие);
- 6) NOT условие;
- 7) условие AND условие;
- 8) условие OR условие.

Эти последние четыре правила определяют новые условия посредством условий. Такое определение (определяющее нечто посредством его самого) называется *рекурсивным определением* и приводит к возможности циклического построения. Например, если условие — это условие, заключенное в круглые скобки, то, повторно применяя это правило, мы получим, что допустимы все следующие условия:

(A-X IS EQUAL TO 5)
 ((A-X IS EQUAL TO 5))
 (((A-X IS EQUAL TO 5)))

Применяя совместно правила (5) и (7), получим условие вида
 (условие AND условие) AND условие

Следовательно, можно было бы подумать, что повторные применения правила (6) приведут к правильному условию вида

NOT NOT NOT A-X IS EQUAL TO 5 (НЕВЕРНО!)

но это не так, потому что в КОБОЛе существует другое правило, которое устанавливает, что слово NOT нельзя использовать более одного раза, не отделяя его круглыми скобками. Таким образом, верным будет следующее условие:

NOT(NOT(NOT A-X IS EQUAL TO 5))

Третья пара круглых скобок вокруг условия A-X IS EQUAL TO 5 и четвертая пара вокруг всего условия допустимы, но не имеют смысла:

(NOT(NOT(NOT(A-X IS EQUAL TO 5))))

Круглые скобки должны быть сбалансированы: число левых и правых скобок должно быть равно точно так же, как в арифметических выражениях. Действительно, если рассматривать знаки логических операций NOT, AND и OR по аналогии со знаками арифметических операций изменения знака, умножения и сложения, то все правила образования скобочных выражений в арифметике применимы для образования составных условий. Обратите внимание, что NOT рассматривается как операция, аналогичная операции изменения знака, а не операции вычитания. В арифметике существует различие между использованием знака «минус» в случаях

—5

и

6—5

В первом случае мы имеем дело со знаком операции изменения знака, т. е. унарным минусом, имеющим только один операнд (5). Во втором случае показан оператор вычитания, или бинарный минус, имеющий два операнда (6 и 5).

Используя правила с (1) по (4) для формирования простых условий и рекурсивно применяя правила с (5) по (8), можно получить весьма сложные составные условия:

NOT(A-X IS EQUAL TO 5 AND B-X IS POSITIVE)
OR(C-X IS NOT NUMERIC AND G-X IS
NOT EQUAL TO 16)

Слово NOT появляется в этом условии три раза, но в двух разных смыслах. В первом вхождении NOT — это логическая операция, которая изменяет значение истинности своего операнда. В двух последующих вхождениях NOT — это часть операции отношения NOT NUMERIC и NOT EQUAL.

Эта двусмысленность может вызвать затруднение, особенно в сокращенных выражениях. Сокращенное условие получается из комбинации условий отношения с логическими операциями NOT, AND и OR. В условии отношения допускается шесть операций отношения, которые уже разбирались ранее. Для обозначения операций отношения разрешается использовать следующие символы:

> для IS GREATER THAN

< для IS LESS THAN

= для IS EQUAL TO

NOT > для IS NOT GREATER THAN

NOT < для IS NOT LESS THAN

NOT = для IS NOT EQUAL TO

За словом NOT, являющимся частью обозначения операций отношения, должен следовать по крайней мере один пробел.

С определенными ограничениями условия отношения, связанные с логическими операциями, можно записывать в сокращенной форме. Условие отношения в общем случае имеет такую форму:

субъект операция-отношения объект

при этом в условии A-X IS EQUAL TO 5 A-X — субъект, IS EQUAL TO — операция отношения и 5 — объект. Субъектом и объектом могут быть любые имена-данных, литеральные значения или арифметические выражения. Последовательность условий отношения может быть соединена логическими операциями, например

A > B AND A > C OR A > D

В этой конкретной последовательности субъекты всех условий отношения одинаковы. Первое правило сокращения условий устанавливает, что общие субъекты могут быть опущены. Тогда приведенный пример можно переписать в виде

A > B AND > C OR > D

В этой последовательности присутствуют общие операции отношения. Они тоже могут быть опущены, в результате чего условие преобразуется к самому короткому виду:

A > B AND C OR D

Второе правило устанавливает, что общие операции отношения (когда имеются общие субъекты) могут быть опущены. Могут быть опущены операции отношения, но не логические операции. Условие

A > B AND A > C AND A > D

нельзя свести к виду

A > B C D

(НЕВЕРНО!)

но можно записать так

A > B AND C AND D

Особую осторожность нужно проявлять при сокращении условий, содержащих NOT, так как в сокращенных условиях NOT всегда интерпретируется как логическая операция и никогда как часть операции отношения. Таким образом,

A > B AND NOT > C

всегда интерпретируется как

$$A > B \text{ AND NOT } A > C$$

Если нужна операция отношения NOT GREATER THAN, то полная последовательность должна быть записана так:

$$A > B \text{ AND } A \text{ NOT } > C$$

В этой последовательности наличие субъекта A и объекта C идентифицирует заключенную между ними конструкцию NOT > как оператор отношения.

Только что описанный метод сокращения применим только к последовательности условий отношения и не может использоваться для других условий. Например, условие

$$A > B \text{ AND IS POSITIVE} \quad (\text{НЕВЕРНО!})$$

неверно и должно быть записано так:

$$A > B \text{ AND } A \text{ IS POSITIVE}$$

Когда последовательность условий записана, независимо от того, сокращена она или нет, може возникнуть неясность, в каком порядке нужно проверять различные условия. В условии NOT T1 AND T2 слово NOT может относиться к группе T1 AND T2 или может относиться только к T1:

$$\text{NOT (T1 AND T2)}$$

или

$$(\text{NOT T1}) \text{ AND T2}$$

где T1 и T2 — это два условия. Один из путей обеспечения ясного и определенного порядка проверки составного условия заключается в использовании круглых скобок, так как главное правило, относящееся к иерархии операций, состоит в том, чтобы всегда проверять отдельно условия, заключенные в круглые скобки. Таким образом, запись

$$((\text{NOT}(\text{NOT T1})) \text{ AND } (\text{T2 OR T3}))$$

не оставляет никаких сомнений, что условия NOT T1 и T2 OR T3 должны проверяться отдельно, приводя на следующем шаге к проверке

$$((\text{NOT T4}) \text{ AND T5})$$

и окончательно порождая

$$(\text{T6 AND T5})$$

Когда скобки опущены, порядок выполнения операций устанавливается согласно старшинству операций, приведенному ниже:

1. класс, имя-условия и знаковые условия;
2. арифметические выражения;
3. условия отношения;
4. NOT;
5. AND;
6. OR.

Рассмотрим для примера условие

$$A * B + 3.5 \text{ NOT } > C + D \text{ AND } E \text{ IS NUMERIC OR NOT } F \text{ EQUAL TO } G$$

Условие *E IS NUMERIC* будет во время исполнения проверяться первым. Заменим его здесь на *T1* (которое может быть истинным или ложным)

$$A * B + 3.5 \text{ NOT } > C + D \text{ AND } T1 \text{ OR NOT } F \text{ EQUAL TO } G$$

Затем вычисляются арифметические выражения. В пределах отдельного выражения умножение и деление будут предшествовать сложению и вычитанию. Используя *X* и *Y* для замещения арифметических выражений, получим условие следующего вида:

$$X \text{ NOT } > Y \text{ AND } T1 \text{ OR NOT } F \text{ EQUAL TO } G$$

Следующими, согласно старшинству операций, проверяются условия отношения. В данном примере их два: *X NOT > Y* и *F EQUAL TO G*. NOT — это часть операции отношения. Замена двух условий отношения на *T2* и *T3* сократит условие до следующего:

$$T2 \text{ AND } T1 \text{ OR NOT } T3$$

Логические операции обрабатываются в порядке их старшинства NOT — AND — OR, приводя к следующим последовательным шагам:

$$T2 \text{ AND } T1 \text{ OR } T4$$

$$T5 \text{ OR } T4$$

$$T6$$

где *T4*, *T5* и *T6* означают результаты выполнения очередных шагов. Все *T* являются условиями, а не именами-данных КОБОЛа. Последнее *T6* имеет только единственное значение: либо истина, либо ложь. Всегда разрешается использовать круглые скобки так, чтобы мог быть получен тот же результат без учета старшинства операций, т. е.

$$(((A * B) + 3.5) \text{ NOT } > (C + D)) \text{ AND } (E \text{ NUMERIC}) \text{ OR } (\text{NOT } (F = G))$$

Упражнения

1. Суммируйте все различные правила и ограничения, применяемые для формирования условий.

2. Расставьте все скобки в следующих условиях:

- а. NOT A GREATER THAN B OR G GREATER THAN H;
- б. A IS ZERO AND $A + B = 4.0$ OR $A \text{ NOT} = Z$;
- в. $A = B$ AND $C = D$ AND $E = F$ OR $G = H$;
- г. NOT NOT NOT A NOT = B.

3. Напишите условие, которое истинно, если пять числовых данных A, B, C, D и E имеют монотонно возрастающие значения, и ложно в противном случае.

4. Напишите условие, которое истинно для всех нечетных значений целочисленного данного и ложно для всех четных значений.

9.5. Модифицируемые передачи управления

Простой и безусловный оператор GO TO (ПЕРЕЙТИ К) имеет формат 1:

GO TO имя-процедуры

где имя-процедуры — это либо имя-параграфа, либо имя-секции. Если используется имя-секции, то не нужно включать слово SECTION, а только само имя. Формат 1 — это форма оператора GO TO, введенная ранее, и во время исполнения он передает управление процедуре, имя которой явно задано. Существует также возможность использовать другой формат оператора GO TO:

формат 2:

GO TO имя-процедуры-1 [имя-процедуры-2]...
DEPENDING ON идентификатор

где идентификатор должен означать элементарное числовое целое данное. (Идентификатор — это имя-данного, которое может быть уточнено и использовано совместно с индексами или именами-индексов.) Во время исполнения при помощи такого оператора GO TO управление будет передаваться одной из указанных процедур в зависимости от текущего значения идентификатора. Управление будет передано по имени-процедуры-1, если идентификатор равен 1, по имени-процедуры-2, если идентификатор равен 2, и т. д., например оператор

GO TO PARAGRAPH-ONE HEAD-SECTION END-RESULT
 DEPENDING ON ITEM-VALUE

обеспечил бы возможность управляемого перехода в программе. Если бы значение данного ITEM-VALUE отличалось от 1, 2 или 3

(в общем случае отличалось бы от значений, начинающихся от 1 и кончающихся числом использованных имен), то оператор GO TO не выполнялся бы, а управление было бы передано следующему оператору. Таким образом, приведенный оператор эквивалентен такой последовательности операторов:

```
IF ITEM-VALUE IS EQUAL TO 1 GO TO PARAGRAPH-ONE.
IF ITEM-VALUE IS EQUAL TO 2 GO TO HEAD-SECTION.
IF ITEM-VALUE IS EQUAL TO 3 GO TO END-RESULT
ELSE NEXT SENTENCE.
```

Оператор GO TO ... DEPENDING ON (ПЕРЕЙТИ К ... В ЗАВИСИМОСТИ ОТ) проще и лаконичнее, чем несколько операторов IF (ЕСЛИ), но приводит к тому же результату. В КОБОЛе, как и в естественном языке, обычно предпочтительна краткость. Формат DEPENDING можно использовать для управления ветвлением программы, передавая управление одной из нескольких процедур в зависимости от типа конкретной записи в файле. Для этого каждая запись должна содержать единственную цифру, идентифицирующую ее тип:

```
01 RECORD-LAYOUT.
```

```
05 CONTROL-VALUE      PICTURE IS 9
                        USAGE IS DISPLAY.
05 REST-OF-RECORD     PICTURE IS X(90).
```

```

.
.
.
```

```
READ INPUT-FILE INTO RECORD-LAYOUT AT END
STOP RUN.
```

```
GO TO
```

```
PROCESS-RECORD-A      PROCESS-RECORD-B
PROCESS-RECORD-C      PROCESS-RECORD-D
DEPENDING ON CONTROL-VALUE.
```

```

.
.
.
```

Другая возможность использования рассматриваемого формата оператора GO TO, состоит в использовании данного, от которого зависит переход, как сигнала, устанавливаемого ранее в программе для управления ветвлением, которое произойдет позже. Еще одна возможность показана на рис. 9.2, на котором представлен программный сегмент для чтения записи, содержащей данные СНА-

РАСТЕР-Х. Это данное может быть любой буквой алфавита и предназначено для управления переходом к одному из двадцати шести ¹⁾ параграфов в зависимости от того, равно ли данное СНА-

WORKING-STORAGE SECTION.

77 DATA-A PICTURE IS 9(2) USAGE IS COMPUTATIONAL
SYNCHRONIZED.

01 TABLE-A.

05 VALUES-OF-LETTERS.

10 FILLER PICTURE IS X VALUE IS "A".

10 FILLER PICTURE IS X VALUE IS "B".

...

10 FILLER PICTURE IS X VALUE IS "Z".

05 LETTER-Q REDEFINES VALUES-OF-LETTERS.

10 LETTER-X PICTURE IS X USAGE IS DISPLAY

OCCURS 26 TIMES

ASCENDING KEY IS LETTER-X

INDEXED BY INDEX-A.

PROCEDURE DIVISION.

...

* ПРЕДПОЛАГАЕТСЯ, ЧТО CHARACTER-X УЖЕ СЧИТАНО.

BRANCH-CONTROL-PARAGRAPH.

SEARCH ALL LETTER-X

WHEN CHARACTER-X IS EQUAL TO LETTER-X (INDEX-A)

SET DATA-A TO INDEX-A

GO TO PROCESS-LETTER-A

PROCESS-LETTER-B

...

PROCESS-LETTER-Z

DEPENDING ON DATA-A.

GO TO NOT-ANY-LETTER-PROCESSING.

Рис. 9.2. Пример использования оператора GO TO..., DEPENDING ON.

РАСТЕР-Х А или В, или С и т. д. Значение CHARACTER-X не-
числовое, так что это данное нельзя использовать в операторе
GO TO ... DEPENDING ON:

GO TO PROCESS-LETTER-A

PROCESS-LETTER-B

...

PROCESS-LETTER-Z

DEPENDING ON CHARACTER-X. (НЕВЕРНО!)

¹⁾ По числу букв в английском алфавите. В русском варианте КОБОЛа в данном случае возможен переход к одному из тридцати шести параграфов.—
Прим. перев.

Значением управляющего данного должно быть обязательно целое положительное число. Осуществить нужное ветвление можно было бы с помощью последовательности из двадцати шести операторов IF, каждый из которых имеет вид:

```
IF CHARACTER-X IS EQUAL TO "A"  
GO TO PROCESS-LETTER-A.
```

но при этом необходима программа, выполняющая по очереди каждый из операторов IF до тех пор, пока не наступит совпадение и переход к требуемому параграфу. Для данного примера это не так страшно, но могут быть ситуации, в которых имеется существенно больше чем двадцать шесть возможностей. В этом случае было бы желательно использовать вариант DEPENDING ON. Это сделано в примере рис. 9.2 с помощью непоследовательного поиска до совпадения среди двадцати шести букв алфавита, хранящихся в таблице LETTER-Q. После этого найденное порядковое значение используется для управления ветвлением в операторе GO TO. Итак, значение данного CHARACTER-X сравнивается с ссылочными значениями в таблице до тех пор, пока не будет найдено подходящее. После этого порядковый номер найденного значения управляет ветвлением. При непоследовательном поиске должно использоваться имя-индекса, а это данное также нечисловое (вспомните, ведь имя-индекса не имеет даже собственной статьи-описания-данного). Преобразование индексного значения в числовое может быть выполнено с помощью оператора SET. Значение данного DATA-A определено как COMPUTATIONAL (ДЛЯ ВЫЧИСЛЕНИЙ), и оператор

```
SET DATA-A TO INDEX-A
```

преобразует относительный машинный адрес, хранящийся в INDEX-A, в порядковый номер 1 или 2, или 3 и т. д., заносимый в DATA-A. Данное DATA-A определено как COMPUTATIONAL SYNCHRONIZED (ДЛЯ ВЫЧИСЛЕНИЙ СДВИНУТО), но оно могло бы быть также специфицировано как DISPLAY (ДЛЯ ВЫДАЧИ). Допустимо и то и другое, если только используется числовой шаблон. TABLE-A — это групповое данное, не имеющее дальнейшего описания, и используется только потому, что фраза OCCURS не может присутствовать на уровне 01. Переопределение VALUES-OF-LETTERS с помощью LETTER-Q позволяет задать в качестве начальных значений элементов таблицы двадцать шесть буквенных литер. Фраза VALUE IS (ЗНАЧЕНИЕ) не может появиться для данных, в описании которых встречается фраза OCCURS (ПОВТОРЯЕТСЯ).

Оператор SEARCH ALL (ИСКАТЬ ОСОБО) вырабатывает значение INDEX-A, для которого проверяемое условие истинно. Это делается не более чем за пять шагов благодаря высокой эффектив-

ности непоследовательного поиска. Последний оператор GO TO NOT-ANY-LETTER-PROCESSING используется в том случае, когда значение CHARACTER-X отличается от буквенного.

Упражнения

1. Запись содержит числовое трехцифровое данное, являющееся управляющим-числом-записи. Напишите ветвящуюся процедуру, которая будет передавать управление различным параграфам в зависимости от того, в какой сотне находится управляющее число.

2. Записи хранятся в четырех различных последовательных файлах на магнитных лентах, образуя числовую последовательность, разбросанную между четырьмя файлами. Это обеспечивается наличием номера-следующего-файла, состоящего из одной цифры, который определяет, в каком из четырех файлов расположена следующая запись последовательности. Это поясняет следующая таблица, в которой первое число — это номер в последовательности, а второе число — это номер-следующего-файла:

Файл 1	Файл 2	Файл 3	Файл 4
100,1	104,4	103,2	105,4
101,1	110,2	107,1	106,3
102,3	111,4	109,2	112,1
108,3			
113,0			

Номера в последовательности записей изменяются от 100 до 113 включительно. У последней записи номер-следующего-файла равен 0, являющемуся признаком конца последовательности. Напишите полную КОБОЛ-программу для вывода файла, в котором все записи расположены в единой числовой последовательности.

3. Напишите процедуру, проверяющую некоторое данное и выполняющую параграфы 1, 2, 5, 7 и 8, если его значение равно единице; выполняющую параграфы 1, 3, 5, 6 и 7, если это значение равно двум, и выполняющую параграфы 2, 4, 5 и 8, если значение данного равно трем.

9.6. Простой оператор PERFORM

Пример процедуры, приведенный на рис. 9.3, содержит различные средства КОБОЛа, которые были рассмотрены ранее. В параграфе P-1 содержится оператор с фразой проверки конца файла и повелительным оператором (в данном случае STOP RUN (ОСТАНОВИТЬ РАБОТУ)). Управляющий оператор STOP RUN завершает выполнение программы и вызывает выход в операционную

систему вычислительной машины. В параграфе P-2 присутствует оператор IF (ЕСЛИ), в котором операторы, выполняющиеся в случае, если условие истинно, сами могут быть условными операторами. Оператор MOVE (ПОМЕСТИТЬ) из этого параграфа служит для

EXAMPLE-X SECTION.

P-1.
READ FILE-A RECORD AT END STOP RUN.

P-2.
IF ITEM-X IS EQUAL TO ZERO
MOVE DATA-VALUE TO DATA-B.

P-3.
GO TO P-6.

P-4.
ADD DATA-A TO DATA-B
ON SIZE ERROR MOVE 999 TO DATA-B.

P-5.
IF DATA-B IS GREATER THAN 3
GO TO P-20.

P-6.
GO TO P-4 P-30 P-40 .
DEPENDING ON DATA-B.

.
.
.

Рис. 9.3. Пример использования средств передачи управления. (Параграфы 20, 30 и 40 опущены для краткости.)

установки значения данного, управляющего в дальнейшем ветвлении в операторе GO TO ... DEPENDING. Оператор IF содержит подразумеваемую фразу ELSE NEXT SENTENCE (ИНАЧЕ СЛЕДУЮЩЕЕ ПРЕДЛОЖЕНИЕ), поэтому управление в любом случае передается параграфу P-3. В этом параграфе безусловный оператор GO TO P-6 обходит следующие два параграфа.

В параграфе P-6 производится ветвление программы, при котором управление может быть передано одному из трех параграфов или секций. Предположим, что управление передается не параграфу P-30 или P-40, а параграфу P-4, в котором арифметический оператор включает необязательную фразу ERROR (ПРИ ПЕРЕПОЛНЕНИИ). В результате выполнения оператора IF пар-

рафа P-5 управление либо передается параграфу P-20, либо параграфу P-6, где снова осуществляется условный переход.

Комбинация этих условий может привести к сложной структуре выполнения программы. Однако существует совершенно иной тип управления, который еще предстоит описать; он часто называется управлением *подчиненными процедурами*¹⁾.

В КОБОЛе процедура представляет собой либо параграф, либо последовательность параграфов, либо целую секцию. Так как параграф составляется из одного или более предложений, то процедура — это просто набор предложений. Но в определении процедуры содержится нечто большее: набор предложений должен выполнять определенную функцию. Процедура может быть проста, как, например:

PARA-ONE.

OPEN INPUT FILE-A.

либо более сложна, как, например, секция EXAMPLE-X на рис. 9.3. В некоторых случаях процедура будет выполняться лишь один раз за время работы программы, например процедура открытия файлов ввода. В других случаях процедура будет выполняться многократно, например при изменении записей файла, до тех пор пока не будет достигнут конец файла. Если процедура предназначена не только для многократного выполнения, но также и для того, чтобы к ней обращались из различных точек программы, то в этом случае организовать управление вычислениями с помощью операторов управления, рассмотренных ранее, можно лишь с большим трудом. Рассмотрим процедуру, вычисляющую контрольную цифру числа и помещающую ее за последней цифрой этого числа. Эта процедура может включать умножение отдельных цифр числа на определенные константы, суммирование получающихся произведений и взятие вычисленной суммы по некоторому модулю. Данная процедура была описана в гл. 5. Если бы возникла необходимость формирования контрольных цифр для различных чисел в различных точках программы, то можно было бы в каждое нужное место программы вставить копию этой процедуры. Но если бы процедура была очень велика или к ней было бы много обращений, то это было бы утомительно и потребовало бы много лишней памяти. Более разумный путь состоит в том, чтобы передавать управление процедуре определения контрольной цифры тогда, когда это необходимо, а затем возвращаться обратно к обычной последовательности выполнения программы до тех пор, пока снова не потребуются эта процедура. Передача управления процедуре определения контрольной цифры может быть реализована довольно просто с помощью

¹⁾ В отечественной литературе подчиненные процедуры обычно называются подпрограммами.— *Прим. ред.*

оператора GO TO CHECK-DIGIT (ПЕРЕЙТИ К КОНТРОЛЬНАЯ-ЦИФРА). Существенная трудность состоит в обеспечении возврата в нужное место после выполнения процедуры. Простой оператор GO TO в конце подчиненной процедуры не сможет это обеспечить, так как возвращаться нужно не в фиксированную точку, а в различные точки для каждого использования процедуры. В этом случае, конечно, мог быть использован оператор GO TO . . . DEPENDING ON, записанный в конце подчиненной процедуры. Тогда непосредственно перед обращением к этой процедуре следовало бы установить значение соответствующего имени-данного, управляющего ветвлением, обеспечив возврат в нужную точку, а именно:

P-1.

```
MOVE 2 TO DATA-ITEM.  
GO TO CHECK-DIGIT.
```

·
·
·

P-8.

```
MOVE 3 TO DATA-ITEM.  
GO TO CHECK-DIGIT.
```

·
·
·

```
STOP RUN.  
CHECK-DIGIT.
```

- * (ВЫПОЛНЕНИЕ НЕОБХОДИМЫХ ВЫЧИСЛЕНИЙ.)
GO TO P1 P8 P12 P15
DEPENDING ON DATA-ITEM.

В этом примере показана только организация передач управления и опущены вычисления или действия других процедур по обработке результатов. Обратите внимание на предложение STOP RUN, которое предохраняет от незапланированного выполнения подчиненной процедуры CHECK-DIGIT в результате «проскока» управления. В КОБОЛе с помощью специального оператора PERFORM (ВЫПОЛНИТЬ), имеющего несколько вариантов, можно вызвать подчиненную процедуру из любого места и автоматически вернуть управление в точку вызывающей процедуры, непосредственно следующую за оператором PERFORM. Оператор PERFORM делает простым использование подчиненных процедур и позволяет программисту писать программу в виде последовательности модулей, которые можно отладить отдельно, а затем связать с помощью

последовательности операторов PERFORM. Этот метод, по существу, аналогичен вызову одной программы из другой, но он применяется только к секциям и параграфам.

Существует несколько вариантов оператора PERFORM. Простейший из них таков:

формат 1:

PERFORM имя-процедуры

где имя-процедуры — это имя параграфа или секции (опять без употребления зарезервированного слова SECTION (СЕКЦИЯ)). Например:

PERFORM CHECK-DIGIT.

В результате выполнения оператора PERFORM управление передается первому оператору упомянутой процедуры. Возврат в вызывающую точку осуществляется после выполнения последнего оператора этой процедуры. Если именем-процедуры является имя-параграфа, то последний оператор параграфа вызывает возврат. Аналогично, если именем-процедуры является имя-секции, то возврат вызывает последний оператор последнего параграфа секции. Возврат могут вызвать только операторы, расположенные в этих конкретных позициях, так что все подчиненные процедуры логически должны завершаться последним написанным оператором. В операторе PERFORM не предусмотрены никакие ссылки на данные. Поэтому все необходимые пересылки данных должны предусматриваться программистом. Например, процедура CHECK-DIGIT, вероятно, обращается к данному в рабочей-памяти, в котором должно храниться исходное число. Засылка значения этого данного и его последующее использование требуют определенных команд, помимо оператора PERFORM. На рис. 9.4 приведен пример использования подчиненной процедуры. Обратите внимание на то, что процедура CHECK-DIGIT расположена не в конце программы (это сделано специально для того, чтобы подчеркнуть, что подчиненные процедуры могут располагаться в любом месте программы) и что один раз она будет выполняться в результате непосредственного перехода, когда управление достигнет параграфа CHECK-DIGIT при обычном последовательном выполнении программы. В данном примере процедура CHECK-DIGIT используется три раза. Прием, использующий возможность исполнения процедуры в результате «проскока» управления, не считается хорошим. К подчиненным процедурам желательно обращаться только с помощью оператора PERFORM.

Ограничение на подчиненные процедуры заключается в том, что их исполнение должно начинаться с первого оператора, следующего за именем-процедуры, а возвращаться в вызывающую точку управление должно только из последнего оператора последнего парагра-

фа. В подчиненной процедуре, имеющей сложный внутренний порядок выполнения команд, подобный приведенному на рис. 9.3, последний записанный оператор может и не быть последним выпол-

WORKING-STORAGE SECTION.

01 DATA-VALUE.

05 NUMBER-X.

10 D-1 PICTURE IS 9.

10 D-2 PICTURE IS 9.

10 D-3 PICTURE IS 9.

05 EXTRA-DIGIT PICTURE IS 9.

...

PROCEDURE DIVISION.

P-1.

OPEN INPUT FILE-A.

READ FILE-A AT END STOP RUN.

P-2.

MOVE ITEM-A TO NUMBER-X.

PERFORM CHECK-DIGIT.

MOVE DATA-VALUE TO OUTPUT-A.

MOVE ITEM-B TO NUMBER-X.

CHECK-DIGIT.

ADD D-1 TO D-3 GIVING RESULT-A.

COMPUTE EXTRA-DIGIT = RESULT-A + 3.0 * D-2.

P-3.

MOVE DATA-VALUE TO OUTPUT-B.

MOVE ITEM-C TO NUMBER-X.

PERFORM CHECK-DIGIT.

MOVE DATA-VALUE TO OUTPUT-C.

STOP RUN.

Рис. 9.4. Пример использования подчиненной процедуры.

няемым оператором. Для этого случая в КОБОЛе предусмотрен пустой оператор. Он таков:

имя-параграфа. EXIT.

Оператор EXIT (ВЫЙТИ) состоит из единственного слова EXIT, за которым следует точка. Он должен быть единственным оператором в параграфе. Ниже приводится пример подчиненной процедуры, использующей оператор EXIT:

SUB-PROCEDURE SECTION.

PARA-1.

IF ITEM-X IS GREATER THAN 50

GO TO PARA-2

ELSE MOVE 250 TO ITEM-Y.
PARA-2.
EXIT.

Параграф EXIT аналогичен параграфу EXIT PROGRAM, но используется он в конце подчиненной процедуры, в то время как EXIT PROGRAM используется в конце подчиненной программы. Уже указывалось, что подчиненная процедура может исполняться подобно другим параграфам программы, если обычная последовательность команд приводит к ее первому оператору без вызова с помощью оператора PERFORM. При этом оператор EXIT не вызывает передачи управления, а управление, минуя его, передается следующему параграфу. Оператор EXIT активизируется в качестве оператора возврата, только когда в подчиненную процедуру вошли посредством оператора PERFORM.

Рассматриваемый до сих пор формат оператора PERFORM позволяет вызывать только единственную подчиненную процедуру. Когда несколько таких процедур записаны непосредственно друг за другом внутри программы, можно исполнять любую последовательность из них за один вызов с помощью следующего варианта оператора PERFORM:

формат 2:

PERFORM имя-процедуры-1
 { THRU
 THROUGH } имя-процедуры-2

При выполнении управление передается первому оператору имени-процедуры-1, а возвращается после выполнения графически последнего оператора имени-процедуры-2. Между двумя этими точками может присутствовать любое число параграфов или секций в обычной последовательности. Любой внутренний оператор EXIT при выполнении обращения игнорируется. Если существует несколько логических путей для достижения конца, то последним параграфом должен быть параграф EXIT, который возвратит управление в исходное место. Ниже приводится пример использования оператора PERFORM для вызова как нескольких параграфов, так и одного:

START-X.
 PERFORM STEP-1 THRU STEP-3.
 PERFORM STEP-1 THRU STEP-2.
 PERFORM STEP-1.
 STOP RUN.
STEP-1.
 ADD A-X TO Z-X.

STEP-2.

ADD B-X TO Z-X.

STEP-3.

ADD C-X TO Z-X.

Заданные выше действия эквивалентны следующим:

ADD A-X TO Z-X.

ADD B-X TO Z-X.

ADD C-X TO Z-X.

ADD A-X TO Z-X.

ADD B-X TO Z-X.

ADD A-X TO Z-X.

При использовании оператора PERFORM подчиненная процедура может содержать любой оператор КОБОЛа, включая и операторы PERFORM. Таким образом, подчиненная процедура может вызывать другую подчиненную процедуру, которая в свою очередь может вызывать еще одну подчиненную процедуру и т. д. Следовательно, процедуры могут быть вложены одна в другую на любую глубину. Единственное ограничение состоит в том, что процедура не может приводить к вызову самой себя ни прямо, ни косвенно через какую-либо из вложенных процедур. Поэтому следующие примеры неверны:

SUB-1.

PERFORM SUB-1. (НЕВЕРНО!)

и

SUB-1.

PERFORM SUB-2.

SUB-2.

PERFORM SUB-1. (НЕВЕРНО!)

Упражнения

1. Напишите процедуру, проверяющую некоторое данное и в зависимости от его значения выполняющую различные параграфы: параграфы с номерами 1, 2, 5, 7 и 8 для значения, равного единице, параграфы с номерами 1, 3, 5, 6 и 7 для значения, равного двум, и параграфы 2, 4, 5 и 8 для значения, равного трем. (Это упражнение уже приводилось в предыдущем разделе, но теперь его следует сделать, используя оператор PERFORM.)

2. Имеются три файла, содержащих записи из 500 литер, в первых трех литерных позициях которых содержится сокращенное название месяца, например JUL (ИЮЛ) или DEC (ДЕК). Напишите

полную КОБОЛ-программу для последовательного чтения записей по очереди из каждого из трех файлов до тех пор, пока не будет найдена запись с месяцем, который принадлежит интервалу времени, начинающемуся за 4 месяца до текущего месяца и оканчивающемуся после 4 месяцев от текущего. После этого распечатайте номер файла, содержащего эту запись.

9.7. Полный оператор PERFORM

Рассмотренные варианты операторов PERFORM обращаются к подчиненной процедуре, выполняющейся только один раз, прежде чем управление возвратится в вызывающую точку. Многократное выполнение подчиненной процедуры может быть задано с помощью зацикливания оператора PERFORM. Например,

P-1.

MOVE 1 TO NUMBER-OF-TIMES.

P-2.

PERFORM INPUT-PROCEDURE.

ADD 1 TO NUMBER-OF-TIMES.

IF NUMBER-OF-TIMES IS NOT GREATER THAN 50
GO TO P-2.

Приведенная выше часть программы приводит к выполнению процедуры INPUT-PROCEDURE пятьдесят раз.

В КОБОЛе предусмотрен вариант оператора PERFORM для более простого задания многократного выполнения подчиненной процедуры.

Формат 3:

PERFORM имя-процедуры-1 [THRU имя-процедуры-2]
 { идентификатор }
 { целое } TIMES

Зарезервированные слова THRU (ПО) и THROUGH (ПО) всегда можно взаимно заменять в программах на КОБОЛе. Приведенный выше пример можно было бы записать так:

PERFORM INPUT-PROCEDURE 50 TIMES.

Идентификатор должен быть числовым элементарным данным, а целое должно быть положительным числом. Если значение повторителя равно нулю или отрицательно, то оператор PERFORM игнорируется. После того как подчиненная процедура будет выполнена указанное число раз, управление передается оператору, следую-

щему за оператором PERFORM. Коль скоро многократное выполнение подчиненной процедуры началось, изменение значения идентификатора в ходе исполнения подчиненной процедуры не будет влиять на число ее повторений. Процедура будет выполнена столько раз, сколько определено исходным значением идентификатора в момент начала вызова. Следовательно, в следующем фрагменте программы:

MOVE 50 TO DATA-VALUE.

PERFORM INPUT-PROCEDURE DATA-VALUE TIMES.

·
·
·

INPUT-PROCEDURE SECTION.

P-1.

ADD 1 TO DATA-VALUE.

·
·
·

P-8.

EXIT.

процедура INPUT-PROCEDURE выполнится точно пятьдесят раз, несмотря на то что окончательное значение DATA-VALUE будет равно 100.

Существует другой способ управления числом исполнений подчиненной процедуры, который состоит в том, что подчиненная процедура выполняется снова и снова до тех пор, пока некоторое условие не станет истинным. Названная возможность задается специальным вариантом оператора PERFORM:

формат 4:

PERFORM имя-процедуры-1 [THRU-имя-процедуры-2]
UNTIL условие

Здесь нет никаких ограничений на условие, как это, например, было в операторах SEARCH. Допускается любое условие, описанное в гл. 4. Пример оператора PERFORM ... UNTIL (ВЫПОЛНИТЬ ... ДО):

PERFORM INPUT-PROCEDURE UNTIL B-X IS EQUAL
TO ZERO.

Если условие истинно в момент начала выполнения оператора PERFORM, то передачи управления подчиненной процедуре не происходит, а выполняется оператор, следующий непосредственно за оператором PERFORM. В случае ложности условия управление передается первому оператору подчиненной процедуры, и такая передача осуществляется каждый раз до тех пор, пока условие не станет истинным. Если условие никогда не станет истинным, то программа заикливается, т. е. подчиненная процедура будет повторяться до тех пор, пока не будет остановлена программа. Следите за тем, чтобы все начальные установки происходили вне подчиненной процедуры, так чтобы такое заикливание не оказалось возможным, как это имеет место в следующем примере:

PERFORM SUB-PROCEDURE UNTIL B-X = 50.

SUB-PROCEDURE SECTION.

P-1.

MOVE 1 TO B-X.

(НЕВЕРНО!)

P-2.

ADD 1 TO B-X.

P-3.

EXIT.

Эта ошибка настолько очевидна, что, казалось бы, нет необходимости предостерегать от нее. Тем не менее подобное заикливание встречается во многих реальных программах.

Более сложные варианты оператора PERFORM допускают регулярное изменение значений от одного до трех данных при повторных исполнениях подчиненной процедуры. Эти варианты часто используются вместе с индексами или индексными-данными для управления обработкой элементов таблиц, определенных фразами OCCURS. Эти варианты оператора PERFORM приводятся ниже.

Формат 5:

<u>PERFORM</u> имя-процедуры-1		[<u>THRU</u> имя-процедуры-2]
<u>VARYING</u>	{ имя-индекса-1 идентификатор-1 идентификатор-3 числовой-литерал-2	<u>FROM</u> { имя-индекса-2 идентификатор-2 числовой-литерал-1
<u>BY</u>		<u>UNTIL</u> условие

Последний формат аналогичен формату 5, за исключением дополнительной фразы AFTER (ЗАТЕМ). Фраза AFTER выполняет почти то же самое, что и фраза VARYING (МЕНЯЯ), по отношению ко второй или третьей величине, однако, ограничения на фразу AFTER несколько отличаются от ограничений на фразу VARYING.

Формат 6:

$$\begin{array}{l}
 \text{PERFORM } \text{имя-процедуры-1} \text{ THRU } \text{имя-процедуры-2} \\
 \text{VARYING} \left\{ \begin{array}{l} \text{имя-индекса-1} \\ \text{идентификатор-1} \end{array} \right\} \text{ FROM } \left\{ \begin{array}{l} \text{имя-индекса-2} \\ \text{идентификатор-2} \\ \text{числовой-литерал-1} \end{array} \right\} \\
 \text{BY} \left\{ \begin{array}{l} \text{идентификатор-3} \\ \text{числовой-литерал-2} \end{array} \right\} \text{ UNTIL } \text{условие-1} \\
 \left[\text{AFTER} \left\{ \begin{array}{l} \text{имя-индекса-3} \\ \text{идентификатор-4} \end{array} \right\} \text{ FROM } \left\{ \begin{array}{l} \text{имя-индекса-4} \\ \text{идентификатор-5} \\ \text{числовой-литерал-3} \end{array} \right\} \right. \\
 \left. \text{BY} \left\{ \begin{array}{l} \text{идентификатор-6} \\ \text{числовой-литерал-4} \end{array} \right\} \text{ UNTIL } \text{условие-2} \quad \dots \right]
 \end{array}$$

Фразу AFTER можно написать один или два раза и не более, ограничивая число имен-индексов или данных, которыми можно управлять, двумя или тремя различными данными. Слово AFTER управляет порядком воздействия на различные данные. Значения числовых-литералов и идентификаторов, следующих непосредственно за словом FROM (ОТ), могут быть любыми числовыми значениями: положительными или отрицательными, нулем или дробью. Величина приращения, определяемая идентификаторами или числовыми литералами, следующими за словом BY (НА), также может быть положительной, отрицательной или дробной, но не может быть нулем. При использовании имен-индексов в качестве начальных порядковых значений и приращений они должны иметь только положительные целые значения. Примеры структур, допускаемых этими форматами оператора PERFORM, приводятся ниже:

PERFORM CHECK-DIGIT

VARYING INDEX-A FROM 1 BY 1

UNTIL INDEX-A IS EQUAL TO 50.

PERFORM OUTPUT-PROCEDURE

VARYING DATA-ITEM-A FROM 10 BY INCREMENT-X

UNTIL OUTPUT-ITEM-X IS NEGATIVE

AFTER SUB-Y FROM 1 BY 1
UNTIL SUB-Y IS GREATER THAN 4.

PERFORM SUB-PROCEDURE-ONE

VARYING SUBSCRIPT-X FROM INDEX-NAME BY 5
UNTIL END-OF-FILE-CONDITION
AFTER DATA-ITEM FROM 100 BY -1
UNTIL DATA-ITEM IS LESS THAN 50
AFTER NAME-X FROM 40 BY 1
UNTIL NAME-X IS GREATER THAN DATA-ITEM.

Сложный оператор PERFORM служит для того, чтобы изменять один или несколько идентификаторов или индексных-данных при повторяющихся исполнениях подчиненной процедуры. Значения идентификаторов должны быть числовыми, в то время как значения имен-индексов должны быть адресные величины. Изменение значений имен-индексов с помощью фраз VARYING и AFTER аналогично действиям, осуществляемым оператором SET, который выполняет взаимные преобразования адресных и порядковых величин. Общая форма обеих фраз VARYING и AFTER такова:

(изменяет) имя-переменной (от) начальное-значение (на)
значение-приращения (пока) условие.

Эффект от наличия одной или двух фраз AFTER заключается в последовательных модификациях двух или трех значений-переменных. Когда изменяется только одно имя-индекса или идентификатор, начальное значение переменной-1 устанавливается перед проверкой условия. Это начальное значение может быть любой числовой величиной. Последовательность выполнения действий показана на рис. 9.5. После установки начального значения проверяется условие. Если условие с самого начала истинно, то управление передается непосредственно оператору, следующему за оператором PERFORM, без выполнения подчиненной процедуры. Если условие ложно, то выполняется подчиненная процедура, и только после ее завершения значение-переменной изменяется на значение-приращения. Это значение-приращения может быть отрицательным или положительным, целым числом или дробным, но оно не может быть нулем. В результате изменения переменная может стать отрицательной, несмотря на то что ее начальное значение не было отрицательным. После этого управление возвращается назад к проверке условия, в результате которой принимается решение, выполнить ли подчиненную процедуру еще раз или перейти к оператору, следующему за оператором PERFORM. Заметьте, что значение переменной изменяется после выполнения подчиненной процедуры. Таким образом, оператор

PERFORM SUB-PROCEDURE
VARYING ITEM-X FROM 1 BY 1
UNTIL ITEM-X IS EQUAL TO 2.

вызовет лишь одно выполнение процедуры SUB-PROCEDURE. Это означает, что программист должен внимательно следить за тем, чтобы процедура повторялась именно желаемое число раз, а не на один раз меньше.

Последовательность выполнения для двух переменных показана на рис. 9.6. Опять же начальные значения переменных установ-

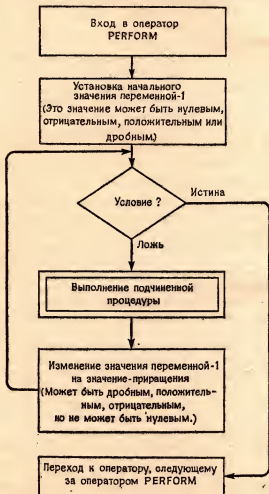


Рис. 9.5. Блок-схема реализации фразы VARYING для одной переменной.

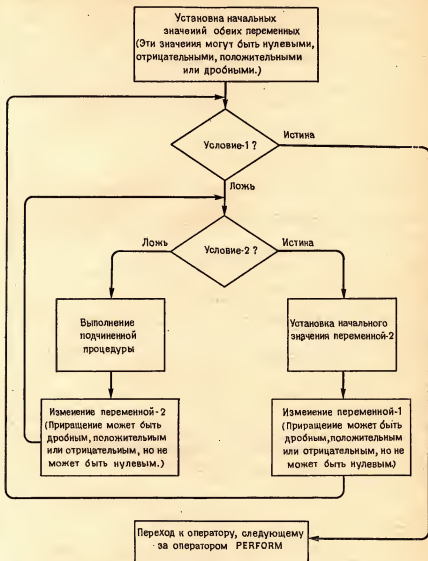


Рис. 9.6. Блок-схема изменения значений двух переменных.

*
*
*

WORKING-STORAGE SECTION.

```
01 A,
   05 B OCCURS 10 TIMES.
      10 C OCCURS 20 TIMES PICTURE IS 9(4).

01 SUBSCRIPT-LIST.
   05 I PICTURE IS 9(3).
   05 J PICTURE IS 9(3).
```

*
*
*

PROCEDURE DIVISION.

```
PERFORM UPDATE-COUNT
  VARYING I FROM 1 BY 1 UNTIL I > 10
  AFTER J FROM 1 BY 1 UNTIL J > 20.
```

UPDATE-COUNT SECTION.

```
P-1.
  ADD 1 TO C (I, J).

P-2.
  EXIT.
```

Эта последовательность операторов добавляет 1 к:

C (1, 1)
C (1, 2)
...
C (1, 20)

а затем к:

C (2, 1)
C (2, 2)
...
C (2, 20)

и т.д. вплоть до: C (10, 20)

Рис. 9.7. Пример использования оператора PERFORM с двумя переменными.

ливаются при входе в оператор PERFORM. В рассматриваемом случае проверяются два условия: сначала условие-1, а затем условие-2. Истинность условия-2 не приводит к выходу из оператора, она вызывает только выход из внутреннего цикла. Во внутреннем цикле выполняется подчиненная процедура и изменяется значение только переменной-2. Если условие-1 не является истинным при проверке, то программа переходит к повторному выполнению подчиненной процедуры, управляемому условием-2, т. е. до тех пор, пока условие-2 не станет истинным. Даже если при этом становится истинным условие-1, это не влияет на производимые действия. Когда управление передается оператору, следующему за оператором PERFORM, значение переменной-2 всегда равно ее начальному значению. Переменная-1 и переменная-2 не должны ссылаться на одно и то же данное ни непосредственно, ни с помощью фраз REDEFINES или RENAMES, но в условиях могут использоваться любые данные. Пример, использующий две переменные, показан на рис. 9.7.

Ситуация для трех переменных аналогична ситуации для двух переменных с той лишь разницей, что в данном случае имеется три цикла. Третья из упомянутых переменных относится к самому внутреннему циклу и изменяется сразу же после выполнения подчиненной процедуры.

Во всех трех случаях изменение величины начального-значения (следующего за словом FROM) в подчиненной процедуре не повлияет на повторную установку начальных значений переменных, но изменение либо самой управляющей переменной, либо значения-приращения повлияет на значение переменной и может также изменить число повторений подчиненной процедуры. Так, оператор PERFORM в следующем примере приведет к заикливанию:

```
PERFORM SUB-PROCEDURE
      VARYING DATA-A FROM 1 BY 2
      UNTIL DATA-A IS EQUAL TO 5.
```

...

SUB-PROCEDURE SECTION.

P-1.

```
      MOVE 1 TO DATA-A.
```

...

P-10.

```
      EXIT.
```

Подчиненная процедура, к которой обращаются с помощью любого из шести форматов оператора PERFORM, как упоминалось

ранее, может содержать любой оператор КОБОЛа, включая другие операторы PERFORM. Процедура, подчиненная подчиненной процедуре, к которой обращаются таким образом, должна быть либо полностью внутри, либо полностью вне соответствующей подчиненной процедуры. Другими словами, не должно быть перекрытия областей вложенных процедур, т. е. должно иметь место либо полное включение области внутренней подчиненной процедуры во внешнюю, как в примере:

```

P-1.
    PERFORM P-2 THRU P-6.
    [ P-2.
      P-3.
        PERFORM P-4 THRU P-5.
      [ P-4.
        P-5.
        P-6.

```

либо области не должны пересекаться вовсе, как в примере, приведенном ниже:

```

P-1
    PERFORM P-2 THRU P-4.
    [ P-2.
      P-3.
        PERFORM P-6 THRU P-7.
      P-4.
      P-5.
      [ P-6.
        P-7.

```

Следовательно, следующая конструкция неверна:

```

P-1.
    PERFORM P-2 THRU P-5.
    [ P-2.
      P-3.
        PERFORM P-4 THRU P-6.
      [ P-4.
        P-5.
        P-6.

```

(НЕВЕРНО!)

В любом случае вложенным процедурам не разрешается иметь общий последний оператор, даже если это параграф EXIT. Поэтому следующая конструкция также неверна:

```

P-1.
  PERFORM P-2 THRU P-5.
  [ P-2.
    P-3.
      PERFORM P-4 THRU P-5.
    [ P-4.
      P-5.
        (НЕВЕРНО!)

```

Упражнения

1. Напишите подчиненную процедуру для чтения точно 300 записей длиной 150 литер из файла ввода. Перепишите в другой файл все те записи, в которых первой литерой является буква Z.

2. Файл содержит записи следующего формата:

01 IN-REC.

```

05 VALUE-X      OCCURS 100 TIMES
                  PICTURE IS 9(5)V9.

```

Напишите подчиненную процедуру, которая будет считывать запись и суммировать сто ее данных, затем будет считывать следующую запись и суммировать значения ее данных с предыдущей суммой и т. д. Запишите в данное RECORD-NUMBER число записей, считая от начала файла, при котором общая сумма данных превысит 40 000.

3. Дана таблица:

01 TWO-WAY-TABLE.

```

05 LINE-ENTRY   OCCURS 300 TIMES.
  10 NAME-OF-ACCOUNT  PICTURE IS A(20).
  10 COST-EACH-MONTH  OCCURS 12 TIMES
    PICTURE IS 9(5)V99.

```

которая полностью сформирована ранее. В ней хранятся 300 наименований счетов и 3600 значений расходов в 300 строках и двенадцати столбцах. Напишите подчиненную процедуру для запоминания в другой таблице

01 ANOTHER-ARRAY.

```

05 EXTRACTED-NAMES  PICTURE IS A (20)
  OCCURS 1 TO 300 TIMES DEPENDING ON SIZE-VALUE.

```

01 SIZE-VALUE PICTURE IS 9(3) VALUE IS ZERO.

наименований-счетов (NAME-OF-ACCOUNT), для которых среднее значение двенадцати ежемесячных-расходов (COST-EACH-MONTH) превышает максимальное значение средних, полученных путем сложения 300 чисел в каждом из двенадцати столбцов и де-

ления сумм на 300. Иными словами, найдите среднее 300 величин для каждого из двенадцати месяцев и выберите наибольшее из этих двенадцати месячных средних для всех счетов. Затем отберите и запомните в таблице ANOTHER-ARRAY все наименования-счетов, чьи годовые средние превышают максимальное среднее по месяцам, полученное по всем счетам.

4. Напишите полную КОБОЛ-программу, которая будет считывать из файла записи

01 CONTROL-RECORD.

05 SELECT-CODE-NUMBER	PICTURE IS 9(4).
05 FILLER	PICTURE IS X(50).
05 TOTAL-AMOUNT-ON-HAND	PICTURE IS X(6).

а затем считывать из второго файла по одной записи

01 INVENTORY-RECORD.

05 VALUES-X OCCURS 100 TIMES INDEXED BY INDEX-A.	
10 HAPHAZARD-CODE-NUMBER	PICTURE IS 9(4).
10 AMOUNT-ON-HAND	PICTURE IS 9(5).

и накапливать сумму всех наличных-сумм (AMOUNT-ON-HAND) для соответствующего номера кода. Соответствующие номера кодов могут быть в каждой записи или их может вообще не быть. В каждой записи эти номера расположены в случайном порядке, так что потребуется последовательный поиск, однако в каждой записи нужное значение может встретиться только один раз. После подсчета суммы всех наличных-сумм для конкретного номера-кода-отбора (SELECT-CODE-NUMBER) занесите эту сумму в данное TOTAL-AMOUNT-ON-HAND, запишите целиком запись CONTROL-RECORD на третий файл вывода и продолжайте чтение записей CONTROL-RECORD, пока соответствующий файл не будет исчерпан.

Глава 10. Непоследовательная обработка файлов

10.1. Устройства массовой памяти

В терминологии КОБОЛа устройство массовой памяти определяется как запоминающая среда, в которой может быть организован как последовательный, так и непоследовательный способ размещения записей. При последовательном способе нельзя получить доступ к произвольной записи (т. е. прочитать ее в память или переписать ее из памяти), не получив предварительно доступ к предыдущей записи. Исключение составляет лишь первая запись файла. Такая последовательная организация записей в файле является характерной для катушек магнитной ленты, колод перфокарт и файлов печати в силу физических особенностей этих устройств. При непоследовательном способе доступ к конкретной записи может быть получен без предварительного получения доступа к какой-либо другой записи файла. К числу запоминающих сред, для которых можно реализовать такой способ организации записей, относятся пакеты магнитных дисков, магнитные барабаны и в некоторой степени специальные ленточные устройства, в которых нет необходимости последовательно читать запись для того, чтобы найти нужную. Бесспорно, самым распространенным примером массовой памяти является пакет дисков, состоящий из нескольких расположенных один над другим отдельных магнитных дисков, напоминающих грампластинки. Общей характеристикой любого устройства массовой памяти является то, что записи хранятся в адресуемых областях, так же как и данные, хранящиеся во внутренней памяти. Задав адрес нужной записи, можно приказывать системе управления массовой памятью, являющейся частью операционной системы вычислительной машины, обратиться непосредственно к области с этим адресом и найти требуемую запись. В силу этого такие устройства часто называются устройствами памяти с прямым доступом.

Время, необходимое для доступа к определенной записи в массовой памяти, является непостоянным. Оно может существенно меняться в зависимости от конкретного физического местоположения записи. Однако, как правило, нет никакой взаимосвязи между по-

зияциями записей в последовательности, в которой они хранятся в файле, и временем доступа, необходимым для их отыскания. Именно поэтому непоследовательный доступ, возможный на устройствах массовой памяти, называется *произвольным* доступом. Время обращения к адресуемой записи, находящейся в массовой памяти, складывается из двух величин: времени поиска, необходимого для отыскания физического положения нужной записи, и времени передачи, необходимого для переписывания записи из устройства с прямым доступом через канал ввода-вывода во внутреннюю память или обратно. Обычно на дисках с плавающими головками время поиска намного превышает время передачи. Для того чтобы найти запись требуется около семидесяти миллисекунд, в то время как для передачи данных во внутреннюю память нужно только около одной миллисекунды. Конечно, на других устройствах эти времена будут другими. Например, для доступа к данным на магнитных барабанах достаточно примерно четырех миллисекунд, а для отыскания записи на дисках с фиксированными головками требуется около двадцати миллисекунд. Но, с другой стороны, сама система управления массовой памятью, используемая в конкретной реализации команд КОБОЛа, может добавить много дополнительного времени сверх времени поиска и передачи. Однако, не приводя точных цифр, можно сказать, что время доступа к записям, хранящимся на устройствах массовой памяти, всегда будет существенно больше, чем время чтения «следующей» записи с магнитной ленты, поскольку при чтении с ленты отсутствует время поиска. Преимущество массовой памяти заключается в том, что возможен непосредственный доступ к нужной записи, в то время как на магнитной ленте должны быть предварительно прочитаны все предшествующие ей записи.

Устройство памяти с прямым доступом физически состоит из ряда дорожек; каждая дорожка содержит ряд областей, в которые заносятся записи. Эти области жестко не фиксируются, и записи различного размера могут храниться на любой дорожке. Но понятие области используется для адресации записи, так что система управления массовой памятью может определить местонахождение записи, сначала отыскав дорожку, а затем и область этой записи. Возможная организация записей, хранящихся на пакете дисков, показана на рис. 10.1. Дорожка представляет собой круговую полосу на поверхности диска, на которой путем намагничивания может быть записана информация. Так как дорожка круговая, то порции информации, непрерывно вращаясь, многократно проходят под головкой чтения-записи, которая будет считывать те данные, которые проходят в текущий момент под ней. Время поиска в этом случае складывается из времени подвода головки чтения-записи к нужной дорожке и времени ожидания прохода нужной области под головкой. Это ожидание может быть либо совсем коротким, либо может потребовать почти полного оборота диска.

Дорожка 1

Запись данных 1	Запись данных 2	Запись данных 3	Промежуток между записями		Запись данных 4	Запись данных 5	Запись данных 6
-----------------------	-----------------------	-----------------------	---------------------------------	--	-----------------------	-----------------------	-----------------------

Дорожка 2

Запись данных 7	Запись данных 8	Запись данных 9	Промежуток между записями		Запись данных 10	Запись данных 11	Запись данных 12
-----------------------	-----------------------	-----------------------	---------------------------------	--	------------------------	------------------------	------------------------

Дорожка 3

Запись данных 13	Запись данных 14	Запись данных 15	Промежуток между записями		Запись данных 16	Запись данных 17	Запись данных 18
------------------------	------------------------	------------------------	---------------------------------	--	------------------------	------------------------	------------------------

Рис. 10.1. Организация файлов на устройствах массовой памяти-1. Пакет дисков может содержать от 10 до 100 областей на дорожке и более чем 2000 дорожек, в зависимости от конкретного устройства:

- Блок 1 расположен на дорожке 1 в области 1.
- Блок 2 расположен на дорожке 1 в области 2.
- Блок 3 расположен на дорожке 2 в области 1.
- Блок 4 расположен на дорожке 2 в области 2.
- Блок 5 расположен на дорожке 3 в области 1.
- Блок 6 расположен на дорожке 3 в области 2.

В приведенном примере записи объединены в блоки в соответствии с фразой КОБОЛА
BLOCK CONTAINS 3 RECORDS.

Последовательная организация

На рис. 10.1 показаны записи, хранящиеся в устройстве массовой памяти при последовательной организации. Способ организации файла задается во время его создания и не может быть изменен в процессе использования файла, т. е. структура файла постоянна. При последовательной организации возможен только последовательный доступ. Работа с такими файлами была впервые описана в гл. 3, а затем снова в гл. 6. Если последовательный файл хранится на устройстве массовой памяти, при его обработке могут быть использованы дополнительные возможности операторов КОБОЛА, которые не упоминались ранее. Во-первых, существуют дополнения к глаголу OPEN (ОТКРЫТЬ), определяющие новый тип передачи данных:

OPEN I-O {имя файла} ...

Полный общий формат оператора OPEN для файлов, хранящихся на устройствах массовой памяти, будет таков:

$$\text{OPEN} \left\{ \begin{array}{l} \text{INPUT} \quad \{\text{имя-файла-1}\} \dots \\ \text{OUTPUT} \quad \{\text{имя-файла-2}\} \dots \\ \text{EXTEND} \quad \{\text{имя-файла-3}\} \dots \\ \text{I-O} \quad \{\text{имя-файла-4}\} \dots \end{array} \right\}$$

Варианты OPEN...REVERSED (ОТКРЫТЬ...РЕВЕРСНО) и OPEN...WITH NO REWIND (ОТКРЫТЬ...БЕЗ ПЕРЕМОТКИ) опущены из этого определения, так как для пакетов дисков они не имеют смысла. Чтение в обратном направлении для дисков отсутствует, поскольку они вращаются с большой скоростью при относительно малой длине дорожки, и система просто ждет в течение оборота, когда запись снова появится под головкой чтения-записи. Отсутствует также и понятие перемотки, так как все записи одинаково доступны и у файла нет физического начала и конца. Однако у файла есть логическое начало, и оператор OPEN устанавливает системный указатель, именуемый указателем текущей записи, таким образом, что он указывает на дорожку и адрес области первой записи файла.

Ранее варианты OPEN INPUT (ОТКРЫТЬ ВХОДНОЙ) и OPEN OUTPUT (ОТКРЫТЬ ВЫХОДНОЙ) описывали односторонний способ передачи данных: для файлов, открытых с помощью варианта INPUT (входных файлов), допускалось использование только операторов READ (ЧИТАТЬ), а для файлов, открытых с помощью вариантов OUTPUT (выходных файлов) или EXTEND (ДОПОЛНЯЕМЫЙ), допускалось использование только операторов WRITE (ПИСАТЬ). Вариант I-O (ВХОДНОЙ-ВЫХОДНОЙ) допускает для одного и того же файла как операции чтения, так и операции записи: записи могут читаться в память и вновь записы-

ваться обратно в файл. Для файлов с последовательной организацией вариант I-O не допускает использования глагола WRITE. Для обратной передачи записи в файл предусмотрен новый глагол REWRITE (ОБНОВИТЬ). Оператор REWRITE имеет формат:

REWRITE имя-записи [FROM идентификатор]

Напомним, что идентификатор — это имя данного, которое может быть уточнено и индексировано с помощью индекса или имени-индекса. Оператор REWRITE заменяет последнюю прочитанную запись в файле, расположенном в массовой памяти, на значение записи, находящейся в области записи, связанной с этим файлом. При использовании оператора REWRITE файл должен быть открыт как I-O файл. Очень важно помнить, что последний оператор ввода-вывода, выполненный для соответствующего файла непосредственно перед оператором REWRITE, должен быть оператором READ. Оператор REWRITE для последовательных файлов может заменять только запись, прочитанную непосредственно перед выполнением этого оператора. При этом перед обновлением могут быть изменены значения данных, содержащихся в этой записи. Именно для этой цели и предусмотрен оператор REWRITE. Процесс обновления, таким образом, состоит из следующих действий: запись читается из файла, изменяется программой, а затем запись в файле заменяется ее новым значением.

Вариант FROM (ИЗ ПОЛЯ) точно так же, как и в случае оператора WRITE, перед обновлением записи помещает данные из поля, определенного идентификатором, в область записи, определенную именем-записи. После обновления записи ее значение в файловой области теряется для программы (точно так же, как для оператора WRITE), если только в параграфе FILE-CONTROL (УПРАВЛЕНИЕ-ФАЙЛАМИ) не указана фраза SAME RECORD AREA (ОБЩАЯ ОБЛАСТЬ ЗАПИСИ). В последнем случае эта запись может быть использована программой в качестве записи для других файлов, упомянутых в этой фразе.

Пример использования оператора REWRITE:

PROCEDURE DIVISION.

P-1.

OPEN I-O INPUT-FILE.

P-2.

READ INPUT-FILE RECORD AT END GO TO P-10.
IF TEST-POSITION IN INPUT-RECORD IS EQUAL
TO "G"

MOVE NEW-VALUES TO RECORD-ITEM-5
 REWRITE INPUT-RECORD.
 GO TO P-2.

...

Файл может быть открыт как I-O (ВХОДНОЙ-ВЫХОДНОЙ) файл, только если он уже существует. Другими словами, он должен быть предварительно создан, для чего он должен быть открыт как OUTPUT (ВЫХОДНОЙ) файл и в него с помощью оператора WRITE должны быть занесены записи. Создание файла может быть произведено в отдельной или в той же самой программе. В последнем случае файл может быть открыт, затем закрыт и снова открыт другим способом, но организация файла, созданного однажды, никогда не может быть изменена. Размеры записей, заносимых в файл, также не должны меняться. Разрешается только менять значения внутри обновляемой записи, но не разрешается увеличивать ее длину. Комбинации способов открытия файлов и глаголов READ, WRITE и REWRITE для файлов с последовательной организацией приведены ниже.

Оператор \ Способ открытия	Организация файла ПОСЛЕДОВАТЕЛЬНАЯ (SEQUENTIAL)			
	I	O	I-O	E
READ	X		X	
WRITE		X		X
REWRITE			X	

(X указывает допустимую комбинацию)

Пример использования I-O файла дан в приведенной ниже программе, которая будет обновлять выбранную запись и запоминать групповое данные с датой и временем (DATE-TIME-GROUP) для фиксации момента обновления. Такой или аналогичный прием фиксации момента обновления очень важен для файлов в массовой памяти, так как если во время выполнения программа неожиданно останавливается, то при отсутствии таких отметок в записях не будет никакой возможности определить, какие из записей были обработаны к моменту остановки программы. В случае файлов на магнитной ленте программа могла бы быть запущена повторно с самого начала, и все записи выходного файла могли бы быть воссозданы.

...

ENVIRONMENT DIVISION.

...

FILE-CONTROL.

SELECT SEQUENTIAL-MASS-STORAGE-FILE
ASSIGN TO DISK
RESERVE 1 AREA
ORGANIZATION IS SEQUENTIAL
ACCESS MODE IS SEQUENTIAL.

...

DATA DIVISION.

FILE SECTION.

FD SEQUENTIAL-MASS-STORAGE-FILE

LABEL RECORDS ARE STANDARD

VALUE OF ID IS "DISKFILE"

DATA RECORD IS MASS-RECORD

RECORD CONTAINS 58 CHARACTERS

BLOCK CONTAINS 1160 CHARACTERS.

01 MASS-RECORD.

05 RECORD-SET-ID

PICTURE IS X(5).

05 DATE-TIME-GROUP.

10 DATE-WRITTEN-X

PICTURE IS 9(5).

10 TIME-WRITTEN-X

PICTURE IS 9(8).

05 RECORD-DATA

PICTURE IS X(40).

...

PROCEDURE DIVISION.

...

OPEN-PARAGRAPH.

OPEN I-O SEQUENTIAL-MASS-STORAGE-FILE.

READ-DATA.

READ SEQUENTIAL-MASS-STORAGE-FILE RECORD
AT END GO TO PARA-5.

IF RECORD-SET-ID IS EQUAL TO SEARCH-ID-VALUE
GO TO REPLACE-PARAGRAPH.

- * ПРЕДПОЛАГАЕТСЯ, ЧТО ДАННОЕ SEARCH-ID-VALUE
 - * БЫЛО СЧИТАНО РАНЕЕ.
- GO TO READ-DATA.

REPLACE-PARAGRAPH.

MOVE NEW-DATA-INFORMATION TO RECORD-DATA.
ACCEPT DATE-WRITTEN-X FROM DAY.
ACCEPT TIME-WRITTEN-X FROM TIME.
REWRITE MASS-RECORD.
GO TO NEXT-STEP.

Упражнения

1. Файл с магнитной ленты нужно переписать в файл на устройстве массовой памяти с прямым доступом. Файл на ленте состоит из записей длиной сто литер каждая, объединенных в блоки по двадцать записей и имеющих следующий формат:

01 TAPE-RECORD.

05 RECORD-NUMBER PICTURE IS 9(6).
05 RECORD-DATA PICTURE IS X(94).

Этот файл на ленте был предварительно отсортирован, так что записи хранятся в порядке увеличения номеров записей (RECORD-NUMBER) от 000001 до 020000. Но в этом файле не более 5000 записей, поэтому часть номеров отсутствует. Ожидается, что в дальнейшем в файл будут заноситься записи с новыми номерами, хотя ни один из них не будет больше, чем 20 000. В случае, когда в последовательности номеров записей входного файла какой-то номер отсутствует, в выходной файл заносится фиктивная запись с этим номером и девяносто четырьмя пробелами. Таким образом, после окончания работы программы файл на устройстве с прямым доступом будет содержать все 20 000 записей. Напишите полную КОБОЛ-программу для копирования файла с ленты и создания файла с последовательной организацией на устройстве с прямым доступом.

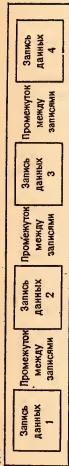
2. Предположим, что файл на устройстве массовой памяти из предыдущего примера уже создан. Напишите программу для чтения с ленты другого файла, имеющего тот же самый формат записей, и обновления записей в файле на устройстве с прямым доступом. Иными словами, замените фиктивные записи записями из файла, читаемого с ленты. Если значение данного RECORD-DATA в записи файла на устройстве с прямым доступом отличается от всех пробелов, то заменять старую запись на новую с тем же номером не нужно, т. е. заменять следует только фиктивные записи. Оба рассматриваемых файла упорядочены по номерам записей.

10.2. Файлы с относительной организацией

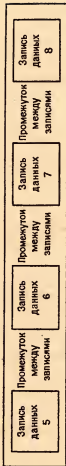
Кроме последовательной допускаются еще два типа организации файлов. Их можно использовать только для файлов, хранящихся на устройствах массовой памяти (обычно на дисковых устройствах). При этом, как и в случае последовательного файла, когда файл уже создан, его организация не может быть изменена. Эти два типа организации сохраняют возможность последовательного доступа к записям (т. е. сохраняют последовательность предшественник — приемник), но они допускают также и прямой доступ к отдельной записи. Этот прямой доступ называется **RANDOM** (ПРОИЗВОЛЬНЫЙ). При использовании последовательного доступа для обеспечения эффективного обновления файла необходимо накапливать достаточное число изменений, которые будут заноситься в файл за один его просмотр. Такое накопление изменений называется обычно группировкой и требует упорядочения входных записей изменений в той же последовательности, в которой упорядочены записи файла. После этого можно последовательно читать эти два набора (набор записей файла и набор записей изменений) и для каждой записи изменений обновлять соответствующую запись файла. С точки зрения использования вычислительной машины такой процесс может быть вполне эффективным, но он может ограничить оперативность системы обработки данных. Например, система предварительных заказов билетов на самолеты не смогла бы функционировать, если бы все заявки на места группировались и обрабатывались бы каждую пятницу, т. е. сортировались по номерам рейсов и на этой основе распределялись места. Вместо этого заявки следует обслуживать в соответствии с правилом, по которому первой обслуживается та заявка, которая первой поступила, и бронирование мест следует выполнять немедленно. Для такого типа обработки файла необходим произвольный (**RANDOM**) доступ.

В КОБОЛе предусмотрены два способа организации произвольного доступа к данным: **RELATIVE** (ОТНОСИТЕЛЬНАЯ) и **INDEX** (ИНДЕКСНАЯ) организация. Сначала будет описана относительная организация, а затем, в следующем разделе — индексная организация. Хотя, конечно, между этими двумя организациями существуют различия, у них есть много общего. Общим является, во-первых, то, что для произвольного доступа необходимо задание некоторого значения, которое будет служить адресом нужной записи. Это значение могло бы быть объединением номеров дорожки и области на дорожке, т. е. могло бы указывать фактический адрес записи на диске. Например, на рис. 10.2, на котором приведен пример относительного файла, запись данных с номером 5 можно найти, указав дорожку с номером 2 и область на дорожке с номером 1. Для программиста такая схема адресации обладает рядом недостатков. Начнем с того, что она сложна. Например, постарайтесь вы-

Дорожка 1



Дорожка 2



Дорожка 3

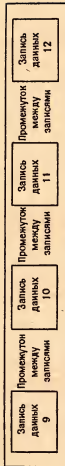


Рис. 10.2. Организация файлов на устройствах массовой памяти-2. С помощью произвольного доступа на нужная запись может быть прочитана в память без предварительного обращения к каким-либо другим записям. Такой файл должен быть создан путем последовательного занесения записей, начиная с области 1 на дорожке 1, при этом доступ к файлу должен быть последовательным:

- Запись 1 расположена на дорожке 1 в области 1.
- Запись 2 расположена на дорожке 1 в области 2.
- Запись 3 расположена на дорожке 1 в области 3.
- Запись 4 расположена на дорожке 1 в области 4.
- Запись 5 расположена на дорожке 2 в области 1.

н т. д.

Объединение записей в блоки для относительных файлов не допускается. Для относительных файлов необходимо задать фразу.

RELATIVE KEY IS имя-данного.

числить, на какой дорожке и в какой области будет найдена запись с номером 1367! Такая адресация также зависит от типа устройства. Различные устройства массовой памяти имеют дорожки разной длины и различную плотность записи, так что фактический адрес данной записи вида дорожка-область будет меняться от устройства к устройству. По этим причинам в языке КОБОЛ вводится понятие *ключа*, дающего программе возможность оперировать с нужной записью при произвольном доступе независимо от типа устройства.

Ключ — это данное КОБОЛа, содержащее значение, которое может быть использовано системой управления массовой памятью для определения местонахождения записи. Для файлов, в описании которых встречается фраза ORGANIZATION IS RELATIVE (ОРГАНИЗАЦИЯ ОТНОСИТЕЛЬНАЯ), ключ — это целое без знака. Программист может идентифицировать запись, помещая числовое значение, например 005, в имя-данного, определенное во фразе

RELATIVE KEY IS имя-данного

Система управления массовой памятью превратит этот номер в адрес дорожки-области; для файла на рис. 10. 2 это были бы дорожка 2 и область 1. В относительном файле записи всегда идентифицируются посредством их относительного расположения в файле: первая запись, вторая запись, третья, четвертая и т. д. Относительный ключ не содержится в записи, он существует только в области рабочей-памяти во внутренней памяти машины. Фразы, описывающие организацию файла, должны находиться в параграфе управления-файлами (FILE-CONTROL) секции ввода-вывода (INPUT-OUTPUT SECTION) раздела оборудования (ENVIRONMENT DIVISION). Для файла с относительной организацией эти фразы должны быть таковы:

ENVIRONMENT DIVISION.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

SELECT имя-файла

ASSIGN TO имя-устройства

RESERVE целое	AREA AREAS
---------------	---------------

ORGANIZATION IS RELATIVE

ACCESS MODE IS	<u>SEQUENTIAL</u> [<u>RELATIVE KEY IS</u> имя-данного-1] <u>RANDOM RELATIVE KEY IS</u> имя-данного-1
FILE STATUS IS имя-данного-2	

Порядок фраз не имеет значения. Имя-данного-1, хранящего относительный ключ (RELATIVE KEY), не может появляться ни в какой статье-описания-записи, связанной с каким-либо файлом. Оно должно быть описано в статье-описания-данного в секции рабочей-памяти (WORKING-STORAGE SECTION) как целое без знака. В этом отношении относительный ключ отличается от ключа, описанного в следующем разделе для файлов с индексной организацией. В случае индексной организации ключ записи должен присутствовать в статье-описания-записи для файла. Все остальные фразы являются необязательными; если ДОСТУП (ACCESS MODE) не определен, то подразумевается, что доступ SEQUENTIAL (ПОСЛЕДОВАТЕЛЬНЫЙ). Рекомендуется включать необязательные фразы для явного описания того, что имеется в виду, а не надеяться на варианты, используемые по умолчанию. Имена-данных могут быть уточнены, но не могут быть индексированы с помощью индексов или имен-индексов. Обратите внимание на то, что всякий раз, когда способ доступа определен как RANDOM (ПРОИЗВОЛЬНЫЙ), должна использоваться фраза RELATIVE KEY, в то время как для последовательного (SEQUENTIAL) доступа эта фраза не обязательна. Не забывайте, что для файла с относительной организацией возможны оба типа доступа. Пример:

FILE-CONTROL.

```

SELECT RELATIVE-MASS-STORAGE-FILE
  ASSIGN TO DISK
  RESERVE 1 AREA
  ORGANIZATION IS RELATIVE
  ACCESS MODE IS RANDOM
  RELATIVE KEY IS RELATIVE-LOCATION-KEY
  FILE STATUS IS STATUS-OF-FILE-CODE.

```

...

WORKING-STORAGE SECTION.

```

77 RELATIVE-LOCATION-KEY PICTURE IS 9(8) USAGE IS
  COMPUTATIONAL SYNCHRONIZED.
01 STATUS-OF-FILE-CODE.
  05 STATUS-KEY-ONE PICTURE IS X USAGE IS DISPLAY.
  05 STATUS-KEY-TWO PICTURE IS X USAGE IS DISPLAY.

```

Организация относительного файла оказывается очень похожей на организацию последовательного файла, но различие заключается в том, что при создании относительного файла система управления массовой памятью устанавливает взаимосвязь между фактическим

физическим адресом дорожки-области и относительным номером, меняющимся от единицы до числа записей, присутствующих в файле. После этого относительный файл можно обрабатывать либо как файл с последовательным доступом, либо как файл с произвольным доступом.

Относительная организация и последовательный доступ

Для создания относительных файлов необходимо использовать последовательный доступ, а сами файлы открывать как выходные. Для этого случая употребляется следующая форма оператора WRITE:

```
WRITE имя-записи  
      [FROM идентификатор]  
      [; INVALID KEY повелительные-операторы]
```

Имя-записи — это одно из имен, описанных с помощью статьи-описания-записи, связанной с относительным файлом. Идентификатор — это имя-данного, которое может быть уточнено и индексировано с помощью индекса или имени-индекса. Фраза INVALID KEY (ПРИ ОШИБКЕ КЛЮЧА) должна ставиться, если для соответствующего имени-файла не определена декларативная секция USE AFTER STANDARD ERROR (ИСПОЛЬЗОВАТЬ ПОСЛЕ СТАНДАРТНОЙ ПРОЦЕДУРЫ ОШИБКИ). Одна из указанных двух возможностей обязательно должна присутствовать. Относительный ключ, указанный в статье SELECT (ДЛЯ) создаваемого относительного файла будет данным в рабочей-памяти. При выполнении первого оператора WRITE запись из файловой области будет переписана на устройство массовой памяти, а в данное, являющееся относительным ключом, системой управления массовой памятью будет занесен номер 1. При выполнении каждого последующего оператора WRITE значение ключа будет увеличиваться на единицу, становясь равным 2, 3, 4 и т. д. Можно использовать это число в качестве счетчика занесенных в файл записей, но никогда не следует менять относительный ключ в процессе создания файла. При последовательном доступе к относительному файлу фраза RELATIVE KEY является необязательной. В этом случае записи в файл заносятся таким же образом, как это делается при наличии фразы RELATIVE KEY, но подсчет их числа во внутренней памяти не производится. И в этом случае можно рекомендовать использовать все необязательные фразы, как это сделано в следующем примере:

ENVIRONMENT DIVISION.

...

* ПРИМЕР СОЗДАНИЯ ОТНОСИТЕЛЬНОГО ФАЙЛА
FILE-CONTROL.

SELECT INPUT-FILE

ASSIGN TO MAGNETIC-TAPE

RESERVE 2 AREAS

ORGANIZATION IS SEQUENTIAL

ACCESS MODE IS SEQUENTIAL.

SELECT RELATIVE-MASS-STORAGE-FILE

ASSIGN TO DISK

RESERVE 1 AREA

ORGANIZATION IS RELATIVE

ACCESS MODE IS SEQUENTIAL

RELATIVE KEY IS KEY-VALUE.

...

FILE-SECTION.

FD INPUT-FILE

LABEL RECORDS ARE STANDARD

VALUE OF ID IS "MG555"

DATA RECORD IS IN-REC

RECORD CONTAINS 100 CHARACTERS

BLOCK CONTAINS 3000 CHARACTERS.

01 IN-REC PICTURE IS X(100).

FD RELATIVE-MASS-STORAGE-FILE

LABEL RECORDS ARE STANDARD

VALUE OF ID IS "RST"

DATA RECORD IS MASS-RECORD

RECORD CONTAINS 100 CHARACTERS.

01 MASS-RECORD PICTURE IS X(100).

WORKING-STORAGE SECTION.

77 KEY-VALUE PICTURE IS 9(8) USAGE IS COMPUTATIONAL
SYNCHRONIZED.

PROCEDURE DIVISION.

MAIN-PROGRAM SECTION.

P-1.

OPEN INPUT INPUT-FILE OUTPUT RELATIVE-MASS-
STORAGE-FILE.

P-2.

READ INPUT-FILE RECORD
AT END GO TO P-10.

WRITE MASS-RECORD FROM IN-REC
INVALID KEY DISPLAY "UNEXPECTED WRITE"
"BEYOND FILE LIMITS"
GO TO P-10.

GO TO P-2.

P-10.

GLOSE INPUT-FILE RELATIVE-MASS-STORAGE-FILE.
STOP RUN.

Ситуация INVALID KEY (ПРИ ОШИБКЕ КЛЮЧА) получает управление при последовательном создании относительного файла, если для очередной выдаваемой записи не хватает места, выделенного на устройстве массовой памяти. Это место определяется в языке управления операционной системы и должно иметь размер, равный или превосходящий размер, необходимый для размещения всех записей нового файла. После того как файл закрыт, его границы устанавливаются в точном соответствии с числом занесенных записей, и в этот файл впоследствии нельзя будет добавить ни одной новой записи. Относительный файл должен создаваться без блокирования записей, хотя в нем допускаются записи переменной длины.

Относительная организация и произвольный доступ

После того как относительный файл создан, записи можно читать и обновлять. Размер записи, заносимой в файл после обновления, не должен превышать размера исходной записи, записанной при создании файла. В статье SELECT такого файла должна быть указана фраза ACCESS IS RANDOM (ДОСТУП ПРОИЗВОЛЬНЫЙ). Следовательно, относительный файл не может быть создан и обновлен в одной и той же программе. Файл в случае обновления следует открывать как I-O файл (ВХОДНОЙ-ВЫХОДНОЙ файл), а для отыскания записи для обновления необходимо использовать оператор READ следующего формата:

READ имя-файла RECORD [INTO идентификатор]
 [: INVALID KEY повелительные-операторы]

Опять же фраза INVALID KEY опускается лишь в том случае, когда для данного файла определена декларативная секция USE AFTER ERROR (ИСПОЛЬЗОВАТЬ ПОСЛЕ ПРОЦЕДУРЫ ОШИБКИ). Пример оператора READ:

READ RELATIVE-MASS-STORAGE-FILE RECORD
 INVALID KEY GO TO ERROR-PARAGRAPH.

Перед выполнением оператора READ необходимо поместить число в данное, представляющее собой относительный ключ. Тогда система управления массовой памятью преобразует значение ключа в адрес дорожки и области, отыщет соответствующую запись и передаст ее значение в область записи во внутренней памяти. Если файл не содержит такой записи, то выполняются либо операторы, указанные во фразе INVALID KEY, либо секция USE AFTER ERROR. Запись может отсутствовать в файле, потому что ранее она была удалена из него с помощью оператора

DELETE имя-файла RECORD
 [: INVALID KEY повелительные-операторы]

Для того чтобы мог быть выполнен оператор DELETE (УДАЛИТЬ), файл должен быть открыт как I-O файл. Запись, которая удаляется из файла и становится недоступной более для программы, задается текущим значением относительного ключа. Если эта запись уже была ранее удалена, возникает ситуация INVALID KEY.

Оператор REWRITE (ОБНОВИТЬ) для произвольного доступа, так же как и другие операторы ввода-вывода, должен содержать фразу INVALID KEY:

REWRITE имя-записи [FROM идентификатор]
 [: INVALID KEY повелительные операторы]

Для произвольного доступа относительный ключ определяет, какая запись в файле должна быть обновлена. Если такой записи в файле нет, активизируется фраза INVALID KEY. В последнем случае запись в файловой области остается доступной для программы. При обновлении файла три оператора READ, DELETE и REWRITE могут быть использованы следующим образом:

P-1.

OPEN INPUT INPUT-FILE
 I-O RELATIVE-MASS-STORAGE-FILE.

P-2.

READ IN-REC AT END GO TO P-10.

MOVE CONTROL-DATA-ITEM OF IN-REC TO KEY-VALUE.

READ RELATIVE-MASS-STORAGE-FILE RECORD
INVALID KEY GO TO P-11.IF TEST-POSITION IN MASS-RECORD IS EQUAL TO "G"
DELETE RELATIVE MASS-STORAGE-FILE RECORD
INVALID KEY GO TO P-11.ELSE MOVE DATA-VALUE-PART IN IN-REC
TO UPDATE-PART OF MASS-RECORD
REWRITE MASS-RECORD

INVALID KEY GO TO P-11.

GO TO P-2.

...

Для относительных файлов допускаются только определенные комбинации операторов ввода-вывода и способов открытия файлов. Эти комбинации показаны в следующей таблице:

Оператор \ Способ открытия	Организация файла ОТНОСИТЕЛЬНАЯ (RELATIVE)			
	I	O	I-O	E
READ	X		X	
WRITE		X	X	
REWRITE			X	
DELETE			X	

(X указывает допустимую комбинацию)

Обратите внимание, что относительный файл не может быть открыт как EXTEND (ДОПОЛНЯЕМЫЙ) файл, т. е. для относительных файлов не допустимы операции дополнения.

10.3. Вычисление относительного ключа

Обновляемая запись, хранящаяся в относительном файле, определяется путем занесения ее номера в относительный ключ. Обычно процесс обновления состоит из считывания записи изменений, затем использования этой записи в качестве основы для чтения записи из относительного файла, изменения считанной записи и занесения ее обратно в файл. Один из путей определения местонахождения записей в относительном файле состоит в использовании некоторого данного в записи изменений для определения значения относительного ключа. Например, значение относительного ключа нужной записи могло бы быть определено с помощью данного EMPLOYEE-IDENTIFICATION-NUMBER (ИДЕНТИФИКАЦИОННЫЙ-НОМЕР-СОТРУДНИКА), если его значение было бы заключено между 0001 и 5000 для сотрудников небольшого учреждения. Если бы запись каждого сотрудника хранилась в файле в относительной позиции, соответствующей идентификационному номеру сотрудника, то данное EMPLOYEE-IDENTIFICATION-NUMBER могло бы быть использовано в качестве аргумента для установки относительного ключа при произвольном чтении, например:

```
READ INPUT-FILE RECORD AT END GO TO P-10.  
MOVE EMPLOYEE-IDENTIFICATION-NUMBER  
  IN TRANSACTION-RECORD TO RELATIVE-KEY-  
  VALUE.  
READ RELATIVE-MASS-STORAGE-FILE RECORD  
  INVALID KEY GO TO P-11.
```

Однако в данном случае необходимо выделить место для хранения 5000 записей, хотя сотрудников может быть и меньше. Если бы использовалось другое данное, такое, как номер карточки социального обеспечения, то было бы невозможно выделить место для 999999999 записей, чтобы учесть все возможные комбинации. В таком случае необходимо осуществить преобразование значения из записи изменений в соответствующее значение относительного ключа. Такое преобразование может быть сделано двумя путями: с помощью поиска в таблице или с помощью некоторого вычислительного метода.

С помощью поиска в таблице можно задать любое преобразование. Этот метод состоит в использовании аргумента из записи изменений для отыскания соответствующего относительного ключа с помощью поиска в таблице связанных пар. Поиск в таблице страдает одним недостатком, связанным с ее размером. В таблице может быть так много строк, что поиск оказывается очень долгим или даже невозможным. Предположим, однако, что в компании работает лишь не-

большое число сотрудников и что компания желает идентифицировать запись каждого сотрудника с помощью номера социального обеспечения. Таблица преобразования могла бы в этом случае выглядеть, например, так:

Номер социального обеспечения	Значение относительного ключа
120-14-5692	144
121-18-0129	58
148-16-7171	14
206-18-6592	892
⋮	⋮
987-65-4321	2

Статьи-описания-данных были бы такими:

01 SOCIAL-SECURITY-TABLE.

05 TABLE-ENTRIES OCCURS 1000 TIMES

ASCENDING KEY IS SOCIAL-SECURITY NUMBER
INDEXED BY INDEX-A.

10 SOCIAL-SECURITY-NUMBER PICTURE IS X(11).

10 LOCATION-KEY-VALUE PICTURE IS 9(8)

USAGE IS COMPUTATIONAL SYNCHRONIZED.

Процедура поиска (в предположении, что значение аргумента, которым является данное SOC-SEC-OF-EMPLOYEE (СОЦ-ОБЕСП-СОТРУДНИКА), уже считано ранее) была бы такова:

SEARCH ALL TABLE-ENTRIES

WHEN SOC-SEC-OF-EMPLOYEE IN TRANSACTION-
RECORD IS EQUAL TO SOCIAL-SECURITY-NUMBER
(INDEX-A) MOVE LOCATION-KEY-VALUE
TO MASS-STORAGE-RELATIVE KEY.

READ RELATIVE-MASS-STORAGE-FILE RECORD
INVALID KEY GO TO P-11.

...

Иногда можно вычислить значение относительного ключа с помощью значения из записи изменений без поиска в таблице. Пример этого был дан ранее в этом разделе, когда значение данного EMPLOYEE-IDENTIFICATION-NUMBER было в точности равно от-

носительной позиции. Вычисление относительного ключа (MASS-STORAGE-RELATIVE-KEY) в этом случае могло бы быть представлено так:

COMPUTE MASS-STORAGE-RELATIVE-KEY
= EMPLOYEE-IDENTIFICATION-NUMBER.

В некоторых случаях вычисления, осуществляющие преобразование, могут быть весьма сложными, но на это стоит пойти, так как преобразование с помощью вычислений выполняется быстрее, чем поиск в таблице. При использовании такого подхода к преобразованию обычно возникает другая проблема. Очень трудно найти такой метод, чтобы результаты вычислений для разных значений аргумента не совпадали. В простом примере, приведенном выше, каждый номер сотрудника (EMPLOYEE-IDENTIFICATION-NUMBER) порождал уникальное значение ключа, которое ссылалось только на одну запись. В общем же случае невозможно отыскать метод вычисления, осуществляющий преобразование, работающее так же однозначно. Но даже в случае неоднозначности вычисления ключа скорость этого метода так велика, что компенсирует недостаток, заключающийся в дублировании значений ключа. В случае неоднозначности вычисления ключа необходимо считывать запись и проверять полное значение аргумента на точное совпадение. Например, рассмотрим компанию со штатом в 10 000 сотрудников. Запись о сотруднике содержит среди других данных следующее:

05 SOC-SEC-NUMBER.

10 FIRST-THREE-DIGITS	PICTURE IS 999.
10 SECOND-TWO-DIGITS	PICTURE IS 99.
10 LAST-FOUR-DIGITS	PICTURE IS 9999.

Пусть вычисления, осуществляющие преобразование, состоят просто в использовании последних четырех цифр (LAST-FOUR-DIGITS) аргумента аналогично тому, как при использовании идентификационного-номера-сотрудника (EMPLOYEE-IDENTIFICATION-NUMBER):

COMPUTE

MASS-STORAGE-RELATIVE-KEY = LAST-FOUR-DIGITS + 1.

Добавление единицы сделано для того, чтобы не было значения относительного ключа, равного нулю. Область возможных результатов заключена между 1 и 10000, что в точности равно числу сотрудников, но маловероятно, чтобы у сотрудников было 10000 различных значений LAST-FOUR-DIGITS. Поэтому программисту следует считывать запись, определенную вычисленным значением относительного ключа MASS-STORAGE-RELATIVE-KEY, и

1) сравнивать номер социального обеспечения в выбранной записи с этим номером в записи изменений; если все девять цифр совпали, то это значит, что выбрана нужная запись;

2) если нет, то следует считывать следующую запись в последовательности (находимую с помощью добавления единицы к относительному ключу), а затем следующую за считанной до тех пор, пока в конечном счете не будет найдено полного совпадения девяти цифр. Это может означать длинный последовательный поиск в файле, но можно надеяться, что дублирование будет происходить не часто и что среднее время поиска будет невелико. Например, после вычисления ключа MASS-STORAGE-RELATIVE-KEY можно применить для поиска следующую процедуру:

P-1.

READ RELATIVE-MASS-STORAGE-FILE RECORD

INVALID KEY GO TO P-11.

IF SOC-SEC-NUMBER IN MASS-RECORD

IS EQUAL TO

SOC-SEC-NUMBER IN TRANSACTION-RECORD

GO TO FURTHER-PROCESSING-PARAGRAPH.

CONTINUE-SEARCH-PARAGRAPH.

ADD 1 TO MASS-STORAGE-RELATIVE-KEY.

IF MASS-STORAGE-RELATIVE-KEY IS NOT GREATER
THAN 10000 GO TO P-1.

...

Последний оператор IF предотвращает последовательный поиск за границами относительного файла. Лучше было бы установить относительный ключ в единицу для продолжения поиска с начала файла. При этом следует установить некоторый признак, для того чтобы этот круговой поиск в файле не продолжался бесконечно.

Пример непоследовательной обработки файла

Целью описанного здесь программного примера является обновление файла инвентаризации (INVENTORY-FILE), т. е. добавление новых записей, изменение значений данных в старых записях и удаление старых записей из файла. Так как относительный файл нельзя расширять после создания, исходный файл необходимо создать достаточно большим для того, чтобы можно было обрабатывать все могущие появиться в дальнейшем данные инвентаризации. Кроме файла инвентаризации, существует еще файл изменений (TRANSACTION-FILE), который содержит записи, вызывающие обновление. Записи в файле изменений расположены в хронологи-

ческом порядке и становятся доступны последовательно. Записи файла инвентаризации доступны в произвольном, или непоследовательном, порядке. Предварительно, до всех обновлений, файл инвентаризации создается с помощью занесения записей, состоящих из всех пробелов, с использованием относительной организации и последовательного доступа. Эти записи служат в качестве фиктивных записей для хранения будущих значений, связанных с некоторым инвентаризируемым товаром. При последующих прогонах программы файл открывается как входной-выходной, так что для данного файла могут выполняться оба оператора READ и REWRITE.

Запись относительного файла инвентаризации содержит шесть полей: код-товара (ITEM-CODE), описатель-товара (ITEM-DESCRIPTOR), наличное-количество (AMOUNT-ON-HAND), последняя-цена-единицы-товара (LATEST-COST-PER-ITEM), заполнитель (FILLER) и код-файла-изменений (UPDATED-BY-TRANS-FILE). Последнее данное указывает на тот файл изменений, который будет использоваться для обновления этой конкретной записи инвентаризации. Его значение состоит из семи цифр: две цифры для года, три цифры для дня в году (от 001 до 365) и две цифры для номера файла (предусматривается, что в один день могут обрабатываться несколько файлов изменений). В общем для этих данных в записи инвентаризации требуется 165 позиций. При последовательном создании файла создаются записи следующего формата:

01 INVENTORY-RECORD.

05 ITEM-CODE PICTURE IS X(9).

05 REST-OF-RECORD PICTURE IS X(156).

При создании файла инвентаризации в него заносится 60000 записей, по одной записи на каждый различный товар, подлежащий инвентаризации. Девятизначный ITEM-CODE (КОД-ТОВАРА) используется для идентификации каждого отдельного инвентаризируемого товара. В фиктивной записи его значение будет устанавливаться равным 999999999. КОБОЛ-программа, приведенная на рис. 10.3, создает файл и заносит в него 60000 фиктивных записей. В секции ESTABLISH-FILE SECTION этой подготовительной программы в ITEM-CODE помещаются все десятки, а в оставшуюся часть записи — все пробелы. В дальнейшем значение из всех девяток будет указывать основной обновляющей программе на фиктивную запись. Без заполнения файла товаров фиктивными записями у обновляющей программы не было бы места для добавления в файл новых записей. Размер относительного файла не может быть увеличен после выполнения оператора CLOSE (ЗАКРЫТЬ) в создающей его программе. Однако путем создания фиктивных записей можно зарезервировать место для занесения реальных записей со значением ITEM-CODE, указывающим на существующий товар.

IDENTIFICATION DIVISION.
PROGRAM-ID. PROG104.

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER. COMPUTER-NAME.
OBJECT-COMPUTER. COMPUTER-NAME.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

SELECT INVENTORY-FILE

ASSIGN TO MASS-STORAGE-DEVICE

ACCESS MODE IS SEQUENTIAL

RELATIVE KEY IS MASS-STORAGE-RELATIVE-KEY

ORGANIZATION IS RELATIVE.

DATA DIVISION.

FILE SECTION.

FD INVENTORY-FILE

LABEL RECORDS ARE STANDARD

RECORD CONTAINS 165 CHARACTERS

DATA RECORD IS INVENTORY-RECORD.

01 INVENTORY-RECORD.

05 ITEM-CODE PICTURE IS 9(9).

05 REST-OF-RECORD PICTURE IS X(156).

WORKING-STORAGE SECTION.

77 MASS-STORAGE-RELATIVE-KEY PICTURE IS 9(8)
USAGE IS COMPUTATIONAL SYNCHRONIZED.

PROCEDURE DIVISION.

ESTABLISH-FILE SECTION.

P-1.

OPEN OUTPUT INVENTORY-FILE.

MOVE 1 TO MASS-STORAGE-RELATIVE-KEY.

P-2.

MOVE 999999999 TO ITEM-CODE IN INVENTORY-RECORD.

MOVE SPACES TO REST-OF-RECORD.

WRITE INVENTORY-RECORD

INVALID KEY

DISPLAY "UNEXPECTED ERROR"

GO TO P-3.

ADD 1 TO MASS-STORAGE-RELATIVE-KEY.

IF MASS-STORAGE-RELATIVE-KEY IS NOT GREATER THAN 60000 GO TO P-2.

P-3.

CLOSE INVENTORY-FILE.

STOP RUN.

Рис. 10.3. Создание файла с произвольным доступом.

Файл изменений имеет одну запись заголовка, которая содержит семизначный идентификационный номер файла изменений (TRANS-FILE-IDENTIFICATION). Этот номер, как было сообщено ранее, состоит из двух цифр года, трех цифр дня и двух цифр номера и используется в процессе обновления, будучи помещен в код-файла-изменений (UPDATED-BY-TRANS-FILE) обновляемой записи инвентаризации. Что-то подобное обязательно требуется при произвольном доступе, так как при прерывании выполнения программы должна быть предусмотрена какая-то возможность запустить программу заново без повторного обновления уже обновленных записей. При каждом требуемом повторном запуске программы подпрограмма, вызванная с помощью оператора CALL, может проверять код-файла-изменений, содержащийся в записи инвентаризации, с тем, чтобы узнать, была ли уже обработана эта запись во время текущего прогона программы. Остальные записи файла изменений содержат код-действия (ACTION-CODE), указывающий, какого типа обработка должна быть выполнена для этой записи: удаление соответствующей записи инвентаризации, модификация значений данных соответствующей записи инвентаризации или добавление рассматриваемой записи в качестве новой записи вместо одной из фиктивных записей. Девятизначный код ITEM-CODE в записи изменений либо указывает на соответствующую запись в файле инвентаризации, либо представляет собой номер, используемый для замены девяти девяток.

Кроме этих данных, запись изменений содержит описатель-товара (ITEM-DESCRIPTOR) — буквенно-цифровое данные, описывающее название инвентаризируемого товара, последнюю-цену-единицы-товара (LATEST-COST-PER-ITEM) и изменение-количества-товара (QUANTITY-MOVED). Последнее данные представляет собой число единиц инвентаризируемого товара, поступивших в запасы или вышедших из запасов. Что конкретно имеется в виду, указывается еще одним данным ADDITION-OR-DELETION (ДОБАВЛЕНИЕ-ИЛИ-УДАЛЕНИЕ).

Основная обновляющая программа для этого примера приведена на рис. 10.4. Она может быть использована для любой обработки файла инвентаризации после его создания. Программа считывает очередную запись изменений и находит соответствующую запись инвентаризации путем выполнения вычислений, осуществляющих преобразование кода-товара (ITEM-CODE) из записи изменений в относительный ключ. Число от 0 до 59999 получается при помощи умножения кода-запаса (SUPPLIER-CODE), содержащегося в коде-товара, на значение-кода (CODE-VALUE), также содержащееся в коде-товара, в результате получается семизначное произведение, остаток от деления которого на 60000 и дает нужное число. К этому числу будет добавлена единица, с тем чтобы избежать нулевого значения. После этого оно будет использовано в качестве

IDENTIFICATION DIVISION.
PROGRAM-ID. PROG105.

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER. COMPUTER-NAME.
OBJECT-COMPUTER. COMPUTER-NAME.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

SELECT TRANSACTION-FILE

ASSIGN TO MAGNETIC-TAPE

RESERVE 2 AREAS

ACCESS MODE IS SEQUENTIAL

ORGANIZATION IS SEQUENTIAL.

SELECT INVENTORY-FILE

ASSIGN TO MASS-STORAGE-DEVICE

ACCESS MODE IS RANDOM

RELATIVE KEY IS MASS-STORAGE-RELATIVE-KEY

ORGANIZATION IS RELATIVE.

DATA DIVISION.

FILE SECTION.

FD TRANSACTION-FILE

LABEL RECORDS ARE STANDARD

RECORD CONTAINS 80 CHARACTERS

BLOCK CONTAINS 800 CHARACTERS

DATA RECORDS ARE TRANSACTION-ID-RECORD TRANSACTION-RECORD.

01 TRANSACTION-ID-RECORD.

05 TRANS-FILE-IDENTIFICATION PICTURE IS 9(7).

05 FILLER PICTURE IS X(73).

01 TRANSACTION-RECORD.

05 ACTION-CODE PICTURE IS 9.

05 ITEM-CODE.

10 MATERIAL-TYPE PICTURE IS 99.

10 SUPPLIER-CODE PICTURE IS 9(3).

10 CODE-VALUE PICTURE IS 9(4).

05 ITEM-DESCRIPTOR PICTURE IS X(25).

05 LATEST-COST-PER-ITEM PICTURE IS X(9).

05 QUANTITY-MOVED PICTURE IS 9(8).

05 ADDITION-OR-DELETION-CODE PICTURE IS X.

88 ADDITION VALUE IS "A".

88 DELETION VALUE IS "D".

05 FILLER PICTURE IS X(27).

FD INVENTORY-FILE

LABEL RECORDS ARE STANDARD

RECORD CONTAINS 165 CHARACTERS

DATA RECORD IS INVENTORY-RECORD.

01 INVENTORY-RECORD.

05 ITEM-CODE PICTURE IS 9(2).

05 ITEM-DESCRIPTOR PICTURE IS X(25).

05 AMOUNT-ON-HAND,	PICTURE IS 9(5).
05 LATEST-COST-PER-ITEM	PICTURE IS \$\$.\$\$\$99.
05 UPDATED-BY-TRANS-FILE	PICTURE IS X(7).
05 FILLER	PICTURE IS X(110).

WORKING-STORAGE SECTION.

77 MASS-STORAGE-RELATIVE-KEY	PICTURE IS 9(8)
USAGE IS COMPUTATIONAL SYNCHRONIZED.	
77 FILE-FLAG	PICTURE IS 9.
77 TEMP-STORAGE-X	PICTURE IS 9(8).
77 TEMP-STORAGE-Y	PICTURE IS 9(8).
77 CURRENT-FILE-CODE	PICTURE IS 9(7).

PROCEDURE DIVISION.

PROCESSING-PROCEDURE SECTION.

OPEN-FILES-PARAGRAPH.

OPEN INPUT TRANSACTION-FILE
I-O INVENTORY-FILE.

READ TRANSACTION-FILE RECORD AT END GO TO P-10.
MOVE TRANS-FILE-IDENTIFICATION TO CURRENT-FILE-CODE.

P-1.

READ TRANSACTION-FILE RECORD AT END GO TO P-10.
MOVE ZERO TO FILE-FLAG.
PERFORM GET-DESIGNATED-RECORD.
GO TO
ADD-NEW-RECORD
UPDATE-CURRENT-RECORD
DELETE-OLD-RECORD
DEPENDING ON ACTION-CODE IN TRANSACTION-RECORD.
DISPLAY "INCORRECT ACTION CODE"
GO TO P-1.

ADD-NEW-RECORD.

IF ITEM-CODE IN INVENTORY-RECORD
IS EQUAL TO 999999999
MOVE CORRESPONDING TRANSACTION-RECORD TO INVENTORY-RECORD.
MOVE QUANTITY-MOVED TO AMOUNT-ON-HAND.
PERFORM WRITE-OUT-INVENTORY-RECORD
GO TO P-1
ELSE
PERFORM SEQUENTIAL-KEY-MODIFICATION,
GO TO ADD-NEW-RECORD.

DELETE-OLD-RECORD.

IF ITEM-CODE IN INVENTORY-RECORD IS EQUAL
TO ITEM-CODE IN TRANSACTION-RECORD
MOVE 999999999 TO ITEM-CODE IN INVENTORY-RECORD
PERFORM WRITE-OUT-INVENTORY-RECORD
GO TO P-1

Рис. 10.4. Обновление относительно файла с произвольным доступом (продолжение).

ELSE
PERFORM SEQUENTIAL-KEY-MODIFICATION
GO TO DELETE-OLD-RECORD.

UPDATE-CURRENT-RECORD. .

IF ITEM-CODE IN INVENTORY-RECORD IS EQUAL
TO ITEM-CODE IN TRANSACTION-RECORD
PERFORM UPDATE-PROCEDURE
PERFORM WRITE-OUT-INVENTORY-RECORD
GO TO P-1

ELSE
PERFORM SEQUENTIAL-KEY-MODIFICATION
GO TO UPDATE-CURRENT-RECORD.

P-10.

PERFORM STOP-EXECUTION

GET-DESIGNATED-RECORD SECTION.

P-1.

MULTIPLY SUPPLIER-CODE BY CODE-VALUE
GIVING TEMP-STORAGE-X.
DIVIDE TEMP-STORAGE-X BY 60000
GIVING TEMP-STORAGE-X
REMAINDER MASS-STORAGE-RELATIVE-KEY.
READ INVENTORY-FILE RECORD
INVALID KEY
DISPLAY "IMPOSSIBLE ERROR"
PERFORM STOP-EXECUTION.

P-2.

EXIT.

WRITE-OUT-INVENTORY-RECORD SECTION.

P-1.

MOVE CURRENT-FILE-CODE TO UPDATED-BY-TRANS-FILE.
REWRITE INVENTORY-RECORD
INVALID KEY
DISPLAY "UNEXPECTED WRITE ERROR"
PERFORM STOP-EXECUTION.

P-2.

EXIT.

SEQUENTIAL-KEY-MODIFICATION SECTION.

P-1.

ADD 1 TO MASS-STORAGE-RELATIVE-KEY.
IF MASS-STORAGE-RELATIVE-KEY IS NOT GREATER
THAN 60000 GO TO P-3.

Рис. 10.4. Обновление относительного файла с произвольным доступом (продолжение).

P-2.

IF FILE-FLAG IS EQUAL TO 1
DISPLAY "HAVE EXCEEDED LIMITS TWICE"
PERFORM STOP-EXECUTION
ELSE
MOVE 1 TO FILE-FLAG
MOVE 1 TO MASS-STORAGE-RELATIVE-KEY.

P-3.

EXIT.

UPDATE-PROCEDURE SECTION.

P-1.

IF ADDITION
ADD QUANTITY-MOVED IN TRANSACTION-RECORD TO AMOUNT-ON-HAND IN
INVENTORY-RECORD.
IF DELETION
SUBTRACT QUANTITY-MOVED IN TRANSACTION-RECORD FROM
AMOUNT-ON-HAND IN INVENTORY-RECORD
ELSE
DISPLAY "IMPROPER ADD OR DELETE CODE"
PERFORM STOP-EXECUTION.
MOVE LATEST-COST-PER-ITEM IN TRANSACTION-RECORD TO
LATEST-COST-PER-ITEM IN INVENTORY-RECORD.

P-2.

EXIT.

STOP-EXECUTION SECTION.

P-1.

CLOSE TRANSACTION-FILE
INVENTORY-FILE.
STOP RUN.

Рис. 10.4. Обновление относительного файла с произвольным доступом (продолжение).

значения относительного ключа для файла на диске. При выборе метода вычислений преследовалась цель, чтобы число одинаковых значений ключа было как можно меньшим. При этом некоторое дублирование значений все же будет иметь место. В результате умножения двух чисел для различных пар аргументов могут встречаться одинаковые результаты. Если произошло дублирование, обновляющая программа начинает поиск в файле инвентаризации до тех пор, пока не будет найдена фиктивная запись или пока не будет обнаружено точное совпадение всех девяти цифр кода-товара. Заметьте, что этот поиск существенно отличается от последовательного, так как он осуществляется путем повторного добавления 1 к относительному ключу и чтения очередной записи из файла. Назовем такой поиск порядковым. Если значение ключа превышает 60000, то он устанавливается в 1 и поиск продолжается с начала файла. Для того чтобы предотвратить заикливание при поиске в файле, в первый раз, когда значение ключа становится больше, чем 60000, устанавливается признак (FILE-FLAG), который проверяется, когда значение ключа в следующий раз становится больше, чем 60000¹⁾.

Упражнения

1. Нарисуйте блок-схему примера программы обновления файла, представленного на рис. 10.4.

2. Составьте таблицу для преобразования аргумента, представляющего собой восьмизначный идентификационный код, хранящийся в записи изменений, в значение относительного ключа для 1000 записей, хранящихся в относительном файле. В предположении, что эта таблица загружена во внутреннюю память с нужными значениями, напишите процедуру выполнения преобразования.

3. Имеется 5000 записей сотрудников, каждая из которых содержит идентификационный-номер-сотрудника, изменяющийся в диапазоне от 05000 до 10000. Напишите процедуру выполнения вычислений, осуществляющих преобразование идентификационного-номера-сотрудника, для определения местонахождения соответствующей записи в файле. (Номер-сотрудника, по которому производится поиск, содержится в файле изменений.)

4. Напишите процедуру выполнения вычислений, осуществляющих преобразование десятизначного аргумента в значение относительного ключа, которое определяет местонахождение одной из 10000 записей, хранящихся на устройстве массовой памяти. Включите шаги, необходимые для нахождения следующей записи в случае, когда значение аргумента не совпадает с соответствующим значением в первой выбранной записи.

¹⁾ Для предотвращения заикливания может быть рекомендован другой более эффективный способ, состоящий в проверке, совпал ли очередной относительный ключ с ключом, с которого мы начали поиск, так как при этом не допускается повторный просмотр записей.— *Прим. ред.*

10.4. Файлы с индексной организацией

Использование относительных файлов страдает определенными недостатками: во-первых, это необходимость преобразования некоторого аргумента в номер относительной позиции n , во-вторых, это невозможность добавить новые записи в существующий файл массовой памяти. Эти ограничения обходятся с помощью другого способа организации файлов, называемого индексной (INDEXED) организацией, которая использует вместо относительного ключа фактический ключ записи. Ключ записи появляется непосредственно в статье-описания-записи для файла и является буквенно-цифровым данным. Это означает, что может быть использовано значение ключа любого типа, а не только числового (как в случае относительного ключа). Когда для индексного файла выполняется оператор ввода-вывода, система управления массовой памятью автоматически превращает значение ключа в номера дорожки и области. Она делает это с помощью стандартной процедуры поиска в таблице точно так же, как в примере, описанном в предыдущем разделе, для преобразования номера социального обеспечения в значение относительного ключа. Программисту при этом не нужно писать процедуру поиска. В случае индексной организации файла поиск осуществляется автоматически системой управления массовой памятью вычислительной машины. Это может осуществляться с помощью таблицы, называемой *индексной*, которая состоит из строк, похожих на приведенные ниже:

Ключ записи (значение, хранящееся также в записи)	Адрес записи на диске	
	дорожка	область
APE	2	3
BAT	4	3
BEE	1	2
CAT	2	2
COW	1	1
DOG	3	1
MULE	3	2
PIG	4	2
RAT	3	3
WASP	2	1
WREN	4	1
YAK	1	3

Пример индексного (INDEXED) файла приведен на рис. 10.5. Индексная таблица может не быть такой детальной, как показанная

в этом примере. Система управления массовой памятью может использовать какую-либо имеющуюся закономерность в расположении записей для уменьшения памяти, отводимой под таблицу. Например, можно воспользоваться тем фактом, что записи, содержащие

Дорожка 1

Ключ 1 "COW"	Данное 1	Ключ 2 "BEE"	Данное 2	Ключ 3 "YAK"	Данное 3
-----------------	----------	-----------------	----------	-----------------	----------

Дорожка 2

Ключ 4 "WASP"	Данное 4	Ключ 5 "CAT"	Данное 5	Ключ 6 "APE"	Данное 6
------------------	----------	-----------------	----------	-----------------	----------

Дорожка 3

Ключ 7 "DOG"	Данное 7	Ключ 8 "MULE"	Данное 8	Ключ 9 "RAT"	Данное 9
-----------------	----------	------------------	----------	-----------------	----------

Дорожка 4

Ключ 10 "WREN"	Данное 10	Ключ 11 "PIG"	Данное 11	Ключ 12 "BAT"	Данное 12
-------------------	-----------	------------------	-----------	------------------	-----------

Рис. 10.5. Организация файлов на устройствах массовой памяти-3. На каждой дорожке расположено по три записи; каждая запись наряду с остальными данными содержит ключ. Для отыскания какой-либо записи необходимо указать значение ее ключа (например, "APE"). Операционная система отыщет в индексной таблице соответствующую строку и обнаружит, что физический адрес искомой записи состоит из номера дорожки 1 и номера области 3. Для индексных файлов необходимо задавать фразу

RECORD KEY IS имя-данного-из-записи.

ключи "DOG" и "MULE", обе расположены на одной дорожке и в соседних областях. Если бы встретилось, например, пятьдесят записей и все на одной дорожке, то система фактически занесла бы в таблицу только строки для первой и последней записей и осуществляла бы интерполяцию для всех промежуточных ключей. Индексная таблица хранится на устройстве массовой памяти вместе с индексным файлом. Она занимает определенное место, и для отыскания в ней адреса дорожки и области требуется некоторое время поиска. В этом заключаются недостатки индексной организации файла, но, если программиста не беспокоят требования места и времени, то

индексная организация оказывается наиболее гибкой и удобной для использования.

Индексный файл описывается специальными фразами в статье SELECT и в соответствующей статье-описания-записи. Статья-описания-записи содержит данное, которое будет использоваться в качестве ключа для доступа к записи. Именно в это данное должны помещаться значения ключа перед тем, как сможет быть выполнен оператор ввода-вывода, обращающийся к индексному файлу. Новые фразы статьи SELECT приведены ниже:

FILE-CONTROL.

```

SELECT имя-файла ASSIGN TO имя-устройства
[ RESERVE целое [ AREA
                  [ AREAS ] ] ]
[ FILE STATUS IS имя-данного-1 ]
ORGANIZATION IS INDEXED
ACCESS MODE IS { SEQUENTIAL
                 RANDOM }
RECORD KEY IS имя-данного-2
[ ALTERNATE RECORD KEY IS имя-данного-3
  WITH DUPLICATES ] ...

```

Например:

FILE-CONTROL.

```

SELECT INDEX-MASS-STORAGE-FILE
ASSIGN TO MASS-STORAGE-DEVICE
RESERVE 1 AREA
ORGANIZATION IS INDEXED
ACCESS MODE IS RANDOM
RECORD KEY IS PRIMARY-KEY-VALUE
ALTERNATE RECORD KEY IS SECONDARY-KEY-
VALUE
ALTERNATE RECORD KEY IS TERTIARY-KEY-
VALUE WITH DUPLICATES
FILE STATUS IS FILE-STATUS-VALUES.

```

Данные для хранения ключей должны быть описаны как буквенно-цифровые в статье-описания-записи, связанной с этим файлом. Этим отличается RECORD KEY (КЛЮЧ ЗАПИСИ) от RELATIVE KEY (ОТНОСИТЕЛЬНЫЙ КЛЮЧ), так как относительный ключ

должен располагаться в рабочей-памяти. Записи могут быть объединены в блоки, но все они должны быть одной длины.

FD INDEX-MASS-STORAGE-FILE

LABEL RECORDS ARE STANDARD

VALUE OF ID IS "XYZ"

RECORD CONTAINS 120 CHARACTERS

DATA ITEM IS INDEXED-RECORD.

01 INDEXED-RECORD.

05 DATA-VALUES-ONE PICTURE IS X(50).

05 TERTIARY-KEY-VALUE PICTURE IS X(10).

05 PRIMARY-KEY-VALUE PICTURE IS X(5).

05 DATA-VALUES-TWO PICTURE IS X(40).

05 SECONDARY-KEY-VALUE PICTURE IS X(15).

Ключи записи могут располагаться в любом месте записи, но если в файле допускаются записи разных типов, т. е. имеется несколько статей-описания-записей, то ключи в записях каждого типа должны занимать одно и то же относительное положение. Значение данного, упомянутого во фразе RECORD KEY (КЛЮЧ ЗАПИСИ), должно всегда определять единственную запись в файле. В отличие от ключа записи значения данных, упомянутых во фразе ALTERNATE RECORD KEY (ДОПОЛНИТЕЛЬНЫЙ КЛЮЧ ЗАПИСИ), могут совпадать для различных записей, но только если для этого ключа задана фраза DUPLICATES (С ДУБЛИРОВАНИЕМ). Набор записей мог бы иметь, например, следующие ключи:

PRIMARY-KEY-VALUE	SECONDARY-KEY-VALUE	TERTIARY-KEY-VALUE
A	1	22
B	5	25
C	9	22
D	8	25
E	7	25
F	4	23
G	3	22

Значения ключа записи хранятся в двух местах: в индексной таблице вместе с адресом дорожки-области и в самой записи вместе с остальными данными. В результате обращения из программы за записью с ключом (PRIMARY-KEY-VALUE) С из файла, при-

веденного в качестве примера выше, будет выбрана третья запись, а при последовательных запросах на поиск записи по значению третичного ключа (TERTIARY-KEY-VALUE), равному 22, сначала была бы найдена первая запись, затем третья запись и, наконец, седьмая запись.

При работе с индексными файлами используются операторы:

```

READ ...WRITE... REWRITE ...
START ... DELETE ...
      OPEN (INPUT/OUTPUT/I-O)...
      CLOSE ...

```

В случае относительных файлов с помощью операторов задаются обращения к записи, определяемой значением относительного ключа. В случае индексных файлов операторы обращаются к записи, определяемой значениями ключей записи, содержащихся в области записи внутренней памяти. Значения ключей могут быть занесены в эту область либо с помощью оператора READ, либо с помощью какого-либо другого оператора КОБОЛа, например оператора MOVE.

Оператор DELETE (УДАЛИТЬ) для индексного файла имеет вид

```

DELETE имя-файла RECORD
[; INVALID KEY повелительные-операторы]

```

Фраза INVALID не должна использоваться для файла, описанного как последовательный. Она должна применяться (если вместо нее не используется секция USE AFTER ERROR) только для файла с произвольным доступом. Для того чтобы мог быть выполнен оператор DELETE, соответствующий файл должен быть открыт как входной-выходной (I-O). Оператор DELETE удаляет запись, определенную текущим значением ключа (RECORD KEY), например:

```

MOVE "ABCDE" TO PRIMARY-VALUE.
DELETE INDEX-MASS-STORAGE-FILE RECORD
      INVALID KEY GO TO P-11.

```

Для приведенного выше примера условие INVALID KEY (ПРИ ОШИБКЕ КЛЮЧА) окажется выполненным, если в файле нет записи с основным ключом, равным "ABCDE".

В зависимости от способа открытия файла над ним могут производиться только определенные действия. В приведенной ниже таблице X помечает допустимые комбинации операторов и способов открытия файла.

Оператор \ Способ открытия			Индексная организация файлов (INDEXED)			
			I	O	I-O	E
Способ доступа	SEQUENTIAL	READ WRITE REWRITE START DELETE	X X	 X	X X X	
	RANDOM	READ WRITE REWRITE START DELETE	X	 X	X X X X	

Заметим, что оператор START может использоваться только для файлов с последовательным доступом, открытых как входные (INPUT) или входные-выходные (I-O). Он не может употребляться для файлов с произвольным доступом. Оператор START (ПОДВЕС-ТИ) предназначен для установки головки чтения-записи на устройстве массовой памяти в положение, обеспечивающее последовательное чтение требуемых записей. Общий формат этого оператора таков:

$$\text{START имя-файла} \left[\text{KEY} \left\{ \begin{array}{l} \text{IS EQUAL TO} \\ \text{IS GREATER THAN} \\ \text{IS NOT LESS THAN} \end{array} \right. \text{имя-данного} \right] \\ \text{[: INVALID KEY повелительные-операторы]}$$

где имя-данного должно быть основным ключом, указанным во фразе RECORD KEY для используемого файла. Необходимо, чтобы в этом операторе была указана фраза INVALID KEY или определена статья USE AFTER ERROR в описании файла, например:

MOVE "ABCDE" TO PRIMARY-KEY-VALUE.

START INDEX-MASS-STORAGE-FILE

KEY IS NOT LESS THAN PRIMARY-VALUE

INVALID KEY GO TO P-11.

В данном примере фраза IS NOT LESS THAN означает, что ищется запись с ключом (KEY), большим или равным значению данного PRIMARY-VALUE. Слово NOT в этой фразе является обязательным, так что допускается только, чтобы искомый ключ был равен,

больше и равен или больше, чем значение имени-данного, указанного в операторе. Если фраза KEY опущена, то по умолчанию в КОБОЛе выбирается отношение EQUAL TO (РАВЕН). Головка чтения-записи по оператору START устанавливается на первую запись, для которой значение хранящегося в ней ключа удовлетворяет заданному в операторе отношению.

Индексный файл с произвольным доступом, открытый как входной-выходной

Индексный файл создается в режиме последовательного доступа, будучи открыт как выходной. После его создания наиболее удобным методом доступа для его обработки является произвольный доступ, а лучшим способом открытия является открытие его как входного-выходного. Это позволяет удалять из него записи, обновлять их и добавлять в файл новые записи с помощью оператора WRITE. Можно выбирать вторичные ключи для определения положения записи дополнительными путями, но основной ключ в каждой записи всегда должен быть уникален.

Рассмотрим пример работы с индексным файлом. Во-первых, создадим индексный файл, используя программу:

FILE-CONTROL.

```
SELECT INDEX-MASS-STORAGE-FILE
  ASSIGN TO DISK
  ORGANIZATION IS INDEXED
  ACCESS MODE IS SEQUENTIAL
  RECORD KEY IS SOCIAL-SECURITY-NUMBER.
```

...

FD INDEX-MASS-STORAGE-FILE

```
  LABEL RECODS ARE STANDARD.
```

01 INDEXED-RECORD.

```
  05 FILLER    PICTURE IS X.
```

```
  05 DATA-AREA.
```

```
    10 SOCIAL-SECURITY-NUMBER
```

```
      PICTURE IS X(9)    USAGE IS DISPLAY.
```

```
    10 FILLER PICTURE IS X(91).
```

...

PROCEDURE DIVISION.

P-1.

```
  OPEN INPUT INPUT-FILE
```

```
    OUTPUT INDEX-MASS-STORAGE-FILE.
```

P-2.

READ INPUT-FILE RECORD INTO DATA-AREA OF
INDEXED-RECORD AT END GO TO P-10.
WRITE INDEXED-RECORD INVALID KEY GO TO P-11.
GO TO P-2.

...

Созданный файл имеет в качестве ключа-записи номер-социального-обеспечения (SOCIAL-SECURITY-NUMBER), так как в программе его создания запись файла INPUT-FILE содержала номер-социального-обеспечения в первых девяти позициях. По оператору WRITE INDEXED-RECORD, запись INDEXED-RECORD отправлялась в устройство массовой памяти, а ее ключ записывался в индексную таблицу для того, чтобы впоследствии эту запись можно было найти. Таким образом, приведенная ниже программа могла бы читать, удалять и добавлять новые записи в рассматриваемый индексный файл:

FILE-CONTROL.

SELECT INDEX-MASS-STORAGE-FILE
ASSIGN TO DISK
ORGANIZATION IS INDEXED
ACCESS MODE IS RANDOM
RECORD KEY IS SOCIAL-SECURITY-NUMBER.
SELECT SEQUENTIAL-TRANSACTION-FILE
ASSIGN TO TAPE
ORGANIZATION IS SEQUENTIAL
ACCESS MODE IS SEQUENTIAL.

...

FD INDEX-MASS-STORAGE-FILE

LABEL RECORDS ARE STANDARD.

01 INDEXED-RECORD.

05 FILLER	PICTURE IS X.
05 SOCIAL-SECURITY-NUMBER	PICTURE IS X(9).
05 DATA-AREA	PICTURE IS X(91).

FD SEQUENTIAL-TRANSACTION-FILE

LABEL RECORDS ARE STANDARD.

01 TRANSACTION-RECORD.

05 SSN

PICTURE IS 9(9).

05 ACTION-CODE

PICTURE IS X.

05 NEW-DATA-VALUE

PICTURE IS X(91).

...

P-1.

OPEN INPUT SEQUENTIAL-TRANSACTION-FILE
I-O INDEX-MASS-STORAGE-FILE.

P-2.

READ SEQUENTIAL-TRANSACTION-FILE-RECORD
AT END GO TO P-10.

* ПРОВЕРКА ДЛЯ ДОБАВЛЕНИЯ НОВОЙ ЗАПИСИ.
IF ACTION-CODE IS "A"

MOVE SSN IN TRANSACTION-RECORD TO
SOCIAL-SECURITY-NUMBER IN INDEXED-RECORD
MOVE NEW-DATA-VALUE TO DATA-AREA
WRITE INDEXED RECORD
INVALID KEY GO TO P-11.

* НОВАЯ ЗАПИСЬ INDEXED-RECORD К ЭТОМУ
* МОМЕНТУ БУДЕТ ЗАНЕСЕНА В ИНДЕКСНЫЙ
* ФАЙЛ, И В ИНДЕКСНУЮ ТАБЛИЦУ
* ФАЙЛА БУДЕТ ЗАПИСАНА СТРОКА, ОТНОСЯЩАЯСЯ
* К ДАННОМУ SOCIAL-SECURITY-NUMBER.
* ПРОВЕРКА ДЛЯ ОБНОВЛЕНИЯ СУЩЕСТВУЮЩЕЙ
* ЗАПИСИ.

IF ACTION-CODE IS "U"

MOVE SSN IN TRANSACTION-RECORD TO
SOCIAL-SECURITY-NUMBER IN INDEXED-RECORD
MOVE NEW-DATA-VALUE TO DATA-AREA
REWRITE INDEXED-RECORD
INVALID KEY GO TO P-11.

* ЗАПИСЬ INDEXED-RECORD БУДЕТ ЗАПИСАНА НА
* МЕСТО УЖЕ СУЩЕСТВУЮЩЕЙ ЗАПИСИ С ТЕМ ЖЕ
* ЗНАЧЕНИЕМ ДАННОГО SOCIAL-SECURITY-NUMBER.
* ПРОВЕРКА ДЛЯ УДАЛЕНИЯ СУЩЕСТВУЮЩЕЙ
* ЗАПИСИ:

IF ACTION-CODE IS "D"

MOVE SSN IN TRANSACTION-RECORD TO
 SOCIAL-SECURITY-NUMBER IN INDEXED-RECORD
 DELETE INDEX-MASS-STORAGE FILE RECORD
 INVALID KEY GO TO P-11.
 GO TO P-2.

10.5. Сортировка

При обработке файлов так часто приходится выполнять сортировку, что в язык КОБОЛ включен целый ряд операторов, помогающих программисту при сортировке файлов. Сортировка состоит в расположении записей в файле в порядке, определяемом возрастанием (убыванием) значений одного или более данных записи. Например, следующие записи отсортированы таким образом, что для одинаковых значений данных ITEM-A, которые сами расположены в возрастающем порядке, значения данных ITEM-B образуют убывающую последовательность:

ITEM-A	ITEM-B	Остаток записи
50	30	...
50	20	...
50	10	...
60	15	...
60	10	...
60	5	...
70	40	...
70	20	...

Данные, определяющие порядок следования записей в файле, называются ключами. Ключи могут быть любыми данными КОБОЛа, как элементарными, так и групповыми, и могут иметь любые стандартные значения данных, будь то числовые, буквенные или буквенно-цифровые. Они не обязательно должны быть расположены в записи слева, как показано в приведенном выше примере; более того, они даже не должны быть смежными. Ключи могут располагаться в любом месте записи. Единственное ограничение заключается в том, что длина и положение ключей должны быть постоянными. В описании ключей не может встречаться фраза OCCURS (ПОВТОРЯЕТСЯ), и они не могут быть подчинены данному, описанному при помощи этой фразы. Размер всех записей в файле должен быть одинаков, а ключи, как уже упоминалось, должны занимать одно и то же положение во всех записях.

Порядок, получающийся в результате операции сортировки, устанавливается путем сравнения значений ключей. При этом значения упорядочиваются следующим образом:

1) для числовых данных: возрастающий порядок совпадает с порядком числовых значений: 5 старше, чем 4; при наличии знака числа возрастающий порядок учитывает и его, так что — 4 старше, чем —5;

2) для буквенных, буквенно-цифровых и редактируемых данных: данные упорядочиваются с помощью политерного сравнения, выполняемого слева направо, так что значение СОТ старше, чем значение САТ. Сравнение производится по тем же правилам, что и в условии отношения.

Сортировка файла производится на ЭВМ в два этапа. Это этап внутренней сортировки и этап слияния. Внутренняя сортировка означает чтение в память некоторого числа записей и упорядочение этих записей с помощью сравнения их ключей. Такое упорядочение требует большого числа сравнений, но для выполнения этих сравнений требуются всего микросекунды. Если бы все записи файла могли быть размещены одновременно во внутренней памяти, то сортировка оказалась бы относительно быстрой процедурой. Но обычно объем данных файла намного превосходит объем внутренней памяти ЭВМ, и записи файла вызываются в память для сортировки порциями. Отсортированные порции заносятся в рабочие файлы. После этого производится слияние порций из рабочих файлов. Затем два (или более) рабочих файла могут быть слиты вместе путем считывания по одной записи и переписи их в требуемом порядке в свободные рабочие файлы. При этом число записей в отсортированных порциях новых рабочих файлов удваивается. Ниже показаны шаги процесса сортировки файла с целыми числами в качестве ключей при наличии очень малой внутренней памяти:

Шаг 1: активизируется исходный сортируемый-файл: 3, 9, 13, 1, 6, 5, 8, 12.

Шаг 2: внутренняя сортировка первой порции (4 записи) приводит к результату: 1, 3, 9, 13.

Шаг 3: первая порция отправляется в рабочий файл 1: 1, 3, 9, 13.

Шаг 4: внутренняя сортировка второй порции (следующие 4 записи) приводит к результату: 5, 6, 8, 12.

Шаг 5: вторая порция записывается в рабочий файл 2: 5, 6, 8, 12.

Шаг 6: рабочие файлы сливаются в файл: 1, 3, 5, 6, 8, 9, 12, 13.

При слиянии сначала будут считаны записи с ключами 1 и 5, и после сравнения запись с ключом 1 будет переписана в сортируемый файл, затем будет считана запись с ключом 3 и она же будет переписана в сортируемый файл, поскольку 3 меньше 5. Затем будет счи-

тана запись с ключом 9. Следующей будет переписана запись с ключом 5 и т. д., пока весь файл не окажется упорядоченным. Так как обычно исходный сортируемый файл содержит больше записей, чем может быть размещено во внутренней памяти за два раза, внутренняя сортировка производится многократно. Затем также много раз будет повторяться слияние, пока не будет получен полностью отсортированный файл. Ниже приведена последовательность действий, используемая в процедуре сортировки КОБОЛа при условии, что записи в исходном сортируемом файле не упорядочены:

1. Выделить как можно больше внутренней памяти под область сортировки.

Фаза сортировки.

2. Загрузить записи в область сортировки, провести внутреннюю сортировку и переписать последовательность упорядоченных записей в рабочий-файл-1. Снова загрузить область сортировки, провести сортировку и переписать последовательность записей на этот раз в рабочий-файл-2.
3. Повторять шаг 2 до тех пор, пока исходный файл не будет исчерпан. В результате образуются два рабочих файла, состоящих из ряда отсортированных последовательностей записей.

Фаза слияния.

4. Выполнить простое слияние первых двух последовательностей с рабочего-файла-1 и рабочего-файла-2, переписывая получающуюся последовательность в рабочий-файл-3. Выполнить то же самое для следующих двух последовательностей записей на рабочих файлах, на этот раз переписывая получающуюся последовательность в рабочий-файл-4. В результате повторных слияний на рабочем-файле-3 и рабочем-файле-4 образуются упорядоченные последовательности записей вдвое более длинные, чем ранее.
5. Перемотать все четыре рабочих файла. Переименовать их так, что теперь рабочие-файлы-3 и -4 будут называться рабочими-файлами-1 и -2 и наоборот. Затем вернуться к шагу 4, который породит упорядоченные последовательности в четыре раза более длинные, чем первоначальные.
6. Продолжать выполнение шагов 4 и 5 до тех пор, пока не будет получена окончательная упорядоченная последовательность, составленная из всех записей. Переписать эту последовательность с того рабочего файла, на котором она окажется, обратно в сортируемый файл. Таким образом, исходный файл будет отсортирован и рабочие файлы больше не нужны.

Изложенный выше вариант алгоритма сортировки является сильным упрощением алгоритмов сортировки, реализованных на ЭВМ. Поскольку сортировка поглощает много машинного времени, авторы программ сортировки ищут разные, подчас довольно сложные пути для ускорения сортировки. Язык КОБОЛ дает программисту возможность довольно просто управлять процессом сортировки независимо от фактического алгоритма реализации сортировки на ЭВМ. Программисту нужно только с помощью одного или двух способов, которые будут описаны позднее, создать сортируемый файл и использовать в разделе процедур оператор SORT (СОРТИРОВАТЬ). Вычислительная система сама будет управлять всеми деталями процесса и завершит работу, когда сортируемый-файл будет упорядочен так, как задано в операторе SORT.

В разделе оборудования сортируемый-файл должен быть включен в статью SELECT (ДЛЯ) вида

SELECT имя-сортируемого-файла ASSIGN TO имя-устройства.

Статья SELECT для сортируемых-файлов имеет приведенный простой вид. Никакие другие фразы, например ACCESS (ДОСТУП) или ORGANIZATION (ОРГАНИЗАЦИЯ), не могут использоваться в этом случае. Для каждого сортируемого-файла, поименованного в параграфе FILE-CONTROL (УПРАВЛЕНИЕ-ФАЙЛАМИ), выделяется и резервируется область-сортировки, необходимая для процесса внутренней сортировки. Если в программе нужно провести несколько сортировок различных файлов, им может быть выделена одна внутренняя область с помощью фраз из параграфа I-O-CONTROL (УПРАВЛЕНИЕ-ВВОДОМ-ВЫВОДОМ):

SAME SORT AREA FOR имя-сортируемого-файла-1 {имя-сортируемого-файла-2...}

При этом в каждый момент выполнения программы может быть открыт только один сортируемый-файл, но это не касается программиста, так как оператор SORT, описанный позднее, автоматически открывает и закрывает сортируемые-файлы, и программист освобождается от забот, связанных с открытием и закрытием этих файлов.

В разделе данных программист должен для каждого имени-сортируемого-файла, указанного в параграфе FILE-CONTROL, написать статью-описания-сортировки с индикатором-уровня SD (ОС). Эта статья аналогична статье FD (ОФ) для обычного файла. Статья SD короче, чем статья FD и имеет вид:

SD имя-сортируемого-файла

[RECORD CONTAINS целое CHARACTERS]
 [DATA { RECORD IS
 RECORDS ARE } { имя-записи } ...]

Обе фразы, следующие за именем-сортируемого-файла, являются обязательными, но их рекомендуется использовать (в любом порядке), так как они позволяют компилятору проверять длину записи и улучшают описание программы. Никакие другие фразы в статье SD не допускаются. Блокировка и вставление меток осуществляются системой обработки файлов конкретной установки. За статьей SD, так же как и за статьей FD, непосредственно следуют одна или более статей-описания-записи. Эти статьи-описания-записи подчиняются всем правилам, установленным для описания записи, и ничем не отличаются от статей, используемых для описания записей в обычных файлах. Для идентификации ключей не предусмотрено никаких специальных обозначений. Ключи указываются в операторе SORT раздела процедур. В сортируемом-файле могут присутствовать записи нескольких типов, но все они должны быть одной длины, и ключи должны занимать одинаковое положение в любой записи каждого типа. Имея ключей в записях различных типов могут совпадать или быть различными. Они могут быть уточнены при обращении посредством оператора SORT.

Ниже приводится пример статей для сортируемого файла:
FILE-CONTROL.

SELECT MY-SORT-FILE ASSIGN TO MAGNETIC-TAPE.

DATA DIVISION.

FILE SECTION.

SD MY-SORT-FILE

RECORD CONTAINS 180 CHARACTERS

DATA RECORD IS SORT-RECORD.

01 SORT-RECORD.

05 EMPLOYEE-LAST-NAME PICTURE IS X(25).

05 EMPLOYEE-FIRST-NAME PICTURE IS X(15).

05 REFERENCE-DATA PICTURE IS X(40).

05 FILLER PICTURE IS X(100).

Все файлы пользователя, в том числе и сортируемый файл, должны будут иметь статьи SELECT и описания. Рабочие файлы совсем не упоминаются в КОБОЛ-программе, но во время работы они выделяются операционной системой. В этом разделе говорилось об использовании четырех рабочих файлов, но на самом деле может использоваться любое число рабочих файлов (от трех и более) в зависимости от того, сколько их может быть выделено для программы. Как правило, чем больше рабочих файлов выделяется, тем быстрее выполняется сортировка.

10.6. Операторы сортировки

Для задания в разделе процедур файловых операций, включающих сортировку, предусмотрено три оператора. Наиболее важным из них является оператор SORT. Другие два оператора RELEASE (ПЕРЕДАТЬ) и RETURN (ВЕРНУТЬ) удобны для подготовки сортируемого-файла и при использовании этого файла после того, как он отсортирован. Существует несколько вариантов оператора SORT. Первый имеет такой формат:

$$\text{SORT имя-сортируемого-файла ON } \left. \begin{array}{l} \text{DESCENDING} \\ \text{ASCENDING} \end{array} \right\} \text{ KEY имя-данного} \\ \text{USING } \{ \text{имя-файла-1} \} \dots \text{GIVING имя-файла-2}$$

Имя-сортируемого-файла должно быть определено в параграфе FILE-CONTROL и описано в статье описания SD. Имя-данного должно быть именем некоторого данного из статьи-описания-записи, следующей за статьей SD. Имя-данного — это ключ, в соответствии со значениями которого будут упорядочиваться записи сортируемого-файла. Кроме сортируемого-файла в операторе указаны еще два файла с именами имя-файла-1 и имя-файла-2, которые должны быть описаны обычным образом в статьях FD (а не SD). Файл с именем имя-файла-1 является входным файлом процедуры сортировки, а файл с именем имя-файла-2 является выходным файлом, в который помещаются отсортированные записи. Обычно статьи-описания-записей для этих файлов будут совпадать со статьями-описания-записей сортируемого-файла. По крайней мере записи должны быть одинакового размера.

Оператор SORT может появляться столько раз, сколько необходимо, в любом месте раздела процедур, за исключением декларативной секции и процедур ввода и вывода, используемых при сортировке (эти процедуры будут описаны позднее в этом разделе). При выполнении оператора SORT автоматически открывается файл с именем имя-файла-1 (который должен быть последовательным файлом) и путем последовательного копирования записей из файла с именем имя-файла-1 в файл с именем имя-сортируемого-файла создается сортируемый-файл. После этого оператор SORT закрывает файл с именем имя-файла-1 и приступает к сортировке сортируемого-файла. Если в декларативных секциях для рассматриваемых файлов присутствуют предложения USE (ИСПОЛЬЗОВАТЬ), которые активировались бы операторами OPEN или CLOSE без каких-либо уточняющих фраз, то автоматическое открытие или закрытие файлов оператором SORT будет активировать эти предложения точно так же, как если бы программист написал операторы OPEN или CLOSE для этих двух последовательных файлов.

Процесс сортировки использует внутреннюю область сортировки

и три или более последовательных рабочих файла, но программисту не нужно беспокоиться об этом. Единственное, о чем он должен позаботиться — это о выделении ресурсов для сортировки. Объем внутренней памяти и число внешних устройств ограничены, и при выполнении программа должна учитывать возможности рабочей машины. После того как процесс сортировки завершен, упорядоченный рабочий файл копируется в сортируемый-файл. Затем осуществляется копирование в файл с именем имя-файла-2 (который также открывается и закрывается автоматически). Файл с именем имя-файла-2 должен быть последовательным. Перед началом сортировки программист должен позаботиться о том, чтобы все три файла были закрыты. После окончания сортировки эти файлы оказываются автоматически закрытыми. Для дальнейшего использования файлов с именами имя-файла-1 и имя-файла-2 необходимо их открыть явно (оператором OPEN). Файл с именем имя-сортируемого файла фактически не доступен для непосредственного использования программистом. Он используется только оператором SORT.

В процессе сортировки записи многократно копируются. Файл с именем имя-файла-1 копируется в сортируемый-файл, который копируется в рабочие файлы. Рабочие файлы многократно копируются друг в друга, а затем один из них (теперь уже упорядоченный) снова копируется в сортируемый-файл. И наконец, сортируемый-файл копируется в файл с именем имя-файла-2. Все файлы закрываются и перематываются, и управление передается оператору, следующему за оператором SORT. Пример простого оператора SORT:

```
SORT MY-SORT-FILE ON ASCENDING KEY
      EMPLOYEE-LAST-NAME
      USING CARD-INPUT-FILE
      GIVING TRANSACTION-DATA-FILE.
```

В формате 1 оператора SORT присутствует только один ключ. На самом деле может быть указано много ключей, и некоторые из них могут быть описаны фразой ON ASCENDING (ПО ВОЗРАСТАНИЮ), в то время как другие могут быть описаны фразой ON DESCENDING (ПО УБЫВАНИЮ). Эти возможности отражены в следующем формате:

SORT имя-сортируемого-файла

ON { DESCENDING
 ASCENDING } KEY имя-данного-1 [имя-данного-2] ...

[ON { DESCENDING
 ASCENDING } KEY имя-данного-3 [имя-данного-4] ...] ...

USING { имя-файла-1 } ...

GIVING имя-файла-2

Последовательность, в которой слева направо упоминаются ключи в операторе SORT, определяет старшинство ключей. По самому старшему ключу сортировка производится первой. Например, оператор

SORT MY-SORT-FILE ON ASCENDING KEY

EMPLOYEE-LAST-NAME EMPLOYEE-FIRST-NAME

породит список упорядоченных фамилий (EMPLOYEE-LAST-NAME), такой, что для одинаковых фамилий имена (EMPLOYEE-FIRST-NAME) также будут упорядочены.

Фразы USING (ИСПОЛЬЗУЯ) и GIVING (ПОЛУЧАЯ) задают процедуре сортировки файл-источник и файл-результат. При копировании по оператору SORT из файла-источника и в файл-результат у программиста нет никакой возможности добавлять или удалять записи или каким-либо образом менять их. Возможность модификации записей в процессе копирования во время выполнения оператора SORT обеспечивается использованием вариантов INPUT PROCEDURE IS (ПРОЦЕДУРА ВВОДА) и OUTPUT PROCEDURE IS (ПРОЦЕДУРА ВЫВОДА) в операторе SORT согласно формату 3:

SORT имя-сортируемого-файла

ON { DESCENDING }
 { ASCENDING } KEY имя-данного

{ USING имя-файла-1
 { INPUT PROCEDURE IS имя-секции-1 [THRU имя-секции-2] }
 { GIVING имя-файла-2
 { OUTPUT PROCEDURE IS имя-секции-3 [THRU имя-секции-4] } }

Обе упомянутые процедуры должны быть секциями. В них нельзя входить обычным образом. Обращения к этим процедурам производятся только при выполнении оператора SORT. Эти секции не должны содержать других операторов SORT и операторов GO TO и PERFORM, обращающихся к операторам, расположенным вне этих секций. Короче говоря, процедуры ввода и вывода, связанные с сортировкой, должны быть изолированы от остальной программы. Вход в них и возврат из них осуществляется только под управлением оператора SORT. Заметьте, что должна присутствовать либо фраза USING, либо фраза INPUT PROCEDURE, а также либо фраза GIVING, либо фраза OUTPUT PROCEDURE, так что существует четыре возможности создания сортируемого-файла и использования отсортированных записей. В случаях USING и GIVING файлы должны быть закрыты. А в случаях INPUT PROCEDURE и OUTPUT PROCEDURE характер использования файлов зависит от программиста, пишущего процедуры.

Когда в операторе SORT используется вариант INPUT PROCEDURE, первым действием оператора SORT является открытие сортируемого-файла. Затем управление передается первому оператору секции с именем имя-секции-1. Управление остается в процедуре ввода до тех пор, пока фактически не выполнится последний ее оператор, т. е. последний оператор в секции с именем имя-секции-1 (или имя-секции-2, если используется вариант THRU). Завершение последнего оператора приводит в действие механизм возврата, который возвращает управление оператору SORT, где сортируемый-файл упорядочивается. Если следующим используется вариант OUTPUT PROCEDURE, то управление передается первому оператору секции с именем имя-секции-3. Предполагается, что процедура вывода будет использовать упорядоченные записи сортируемого-файла. Управление остается в процедуре вывода до тех пор, пока не выполнится ее последний оператор. После этого управление передается снова в оператор SORT. При этом закрывается сортируемый-файл. Тогда (и только тогда) управление передается оператору, следующему за оператором SORT. В любой момент выполнения оператора SORT может возникнуть аварийная ситуация и активизироваться декларативная секция. Однако эта возможность пока еще не описана. Предложение USE (ИСПОЛЬЗОВАТЬ) для ошибок ввода или вывода описывается в следующей главе.

В процедуре ввода, используемой при сортировке, по крайней мере один раз должен встретиться оператор RELEASE (ПЕРЕДАТЬ). Назначение этого оператора аналогично назначению оператора WRITE: он используется для передачи записи из области записи, связанной с сортируемым-файлом, в этот файл. Его формат таков:

RELEASE имя-сортируемого-файла [FROM имя-записи]

Если используется вариант FROM (ИЗ ПОЛЯ), то прежде, чем содержимое сортируемой записи будет переписано в сортируемый-файл, значение записи с именем имя-записи будет помещено в сортируемую запись. Это перемещение осуществляется без варианта CORRESPONDING (СООТВЕТСТВЕННО). Оператор RELEASE — единственный оператор вывода, который может быть использован для занесения записи в сортируемый-файл. Примером процедуры ввода могла бы быть следующая процедура:

PREPARE-SORT-FILE SECTION.

PARA-ONE.

OPEN INPUT CARD-INPUT-FILE.

PARA-TWO.

READ CARD-INPUT-FILE AT END GO TO PARA-THREE.

IF CARD-CONTROL-CODE IN CARD-RECORD IS EQUAL

TO "SKIP" GO TO PARA-TWO
 ELSE RELEASE MY-SORT-RECORD FROM CARD-
 RECORD GO TO PARA-TWO.

PARA-THREE.

CLOSE CARD-INPUT-FILE.

В процедуре вывода, используемой при сортировке, по крайней мере один раз должен встретиться оператор RETURN (ВЕРНУТЬ). Этот оператор выполняет для сортируемого файла те же действия, которые оператор READ выполняет для обычного последовательного файла: он читает следующую запись из сортируемого-файла в область сортируемой записи. Его формат таков:

RETURN имя-сортируемого-файла RECORD INTO идентификатор
AT END повелительные-операторы

Фраза INTO (В) действует так же, как и оператор MOVE: она задает перемещение сортируемой записи в некоторую другую область внутренней памяти. Если с идентификатором связан индекс или имя-индекса, то индексирование осуществляется после того, как запись считана и перед выполнением перемещения. Оператор RETURN — это единственный оператор ввода, который может быть использован для сортируемого-файла. Операторы RETURN и RELEASE нельзя использовать в одной и той же процедуре, так как оператор RELEASE заносит запись в сортируемый-файл до сортировки, а оператор RETURN читает записи обратно в память после завершения сортировки.

10.7. Оператор MERGE

Оператор MERGE (СЛИТЬ) совершенно не связан с оператором SORT, хотя отмечалось, что при выполнении оператора SORT сначала производится внутренняя сортировка, а затем слияние. Однако слияние при сортировке применяется только к рабочим файлам и не может управляться программистом. Оператор MERGE введен специально для того, чтобы дать программисту возможность объединять файлы, которые уже были отсортированы в соответствии с определенными ключами. Формат этого оператора аналогичен формату оператора SORT:

MERGE имя-файла-слияния

ON { DESCENDING } KEY [имя-данного-1] ...

[ON { DESCENDING } KEY [имя-данного-2] ...] ...

USING имя-файла-1 [имя-файла-2] ...

{ GIVING имя-файла-3

OUTPUT PROCEDURE IS имя-секции-1 [{ THRU } THROUGH] имя-секции-2 }

Имя-файла-слияния должно быть описано в статье с индикатором-уровня SD (OC) раздела данных и в статье SELECT (ДЛЯ) параграфа FILE-CONTROL раздела оборудования. Так же как и сортируемый-файл, файл-слияния является «посторонним файлом», поскольку программист не может использовать его непосредственно. Файлы, указанные во фразе USING, сливаются вместе и копируются в файл-слияния. После этого файл-слияния перематывается, и его содержимое становится доступным посредством вариантов GIVING или OUTPUT PROCEDURE.

Для файлов с именами имя-файла-1, имя-файла-2 и т. д. области-записей должны быть описаны в статье FD, а не в статье SD. Они, так же как и все упомянутые здесь файлы, должны появиться в статье SELECT. Рабочие файлы в операторе MERGE не используются. Все записи во всех файлах должны быть одной и той же длины, но коэффициент блокировки для любого сливаемого файла может быть свой. Эти входные (сливаемые) файлы не должны быть открыты к моменту выполнения оператора MERGE. Оператор MERGE сам откроет, а при завершении сам закроет их (фактически выполняется оператор CLOSE без необязательных вариантов). Во время обработки файлов, указанных в операторе, будут автоматически выполняться соответствующие декларативные секции, если такие определены. Другие ограничения заключаются в том, что если на одной катушке расположено несколько файлов, то имя только одного из них может употребляться в операторе MERGE, и в том, что в этом операторе имена-файлов не могут повторяться. В одной программе можно использовать несколько операторов MERGE, однако эти операторы не должны появляться в процедурах ввода или вывода, связанных с сортировкой или слиянием, и в декларативных секциях.

Имена-данных-1, -2 и т. д. являются именами данных из статьи-описания-записи файла-слияния. Их можно уточнять, но нельзя индексировать с помощью индексов или имен-индексов. Они записываются слева направо в порядке уменьшения их старшинства аналогично тому, как записываются ключи в операторе SORT.

Сливаемые файлы должны быть ранее отсортированы в том порядке, который определен в операторе MERGE. Если порядок записей не таков, слияние будет выполнено неверно. Если при слиянии значения ключей в каких-то записях совпадают, то эти записи заносятся в файл-слияния в том порядке, в котором в операторе MERGE записаны имена соответствующих файлов, т. е. имя-файла-1, имя-файла-2 и т. д.

В случае если в операторе указан вариант GIVING, результат слияния будет скопирован из файла-слияния в файл с именем имя-файла-3. Если же указан вариант OUTPUT PROCEDURE, то результат слияния становится доступен для обработки, определенной в указанной секции (или секциях). Заметьте, что процедуры вывода должны быть секциями, а не параграфами. Эти секции должны быть расположены в разделе процедур таким образом, чтобы они не могли быть выполнены путем обычной передачи управления. Входить в них можно только из оператора MERGE. Процедура вывода должна содержать по крайней мере один оператор RETURN (ВЕРНУТЬ) для чтения записей из файла-слияния во внутреннюю память и будет обычно содержать оператор EXIT (ВЫЙТИ) для передачи управления обратно оператору MERGE. Формально управление возвращается в оператор MERGE после выполнения оператора, написанного в процедуре вывода последним, но при наличии сложного ветвления в процедуре вывода обычно бывает удобнее осуществлять возврат с помощью оператора EXIT, который может быть помещен в любое место секции. Напомним, что оператор EXIT должен быть единственным предложением в отдельном параграфе. Если кроме файла-слияния в процедуре вывода используются какие-либо другие файлы, то их имена не должны быть указаны в операторе MERGE. Эти файлы должны быть открыты и закрыты самим программистом и не подлежат управлению со стороны оператора MERGE.

Последовательность операций, выполняемых оператором MERGE, приводится ниже.

1. Управление передается оператору MERGE.
2. Файл-слияния, файлы с именами имя-файла-1, имя-файла-2 и все остальные сливаемые файлы открываются и устанавливаются в исходное положение. Выполняются все действия по обработке меток-пользователя, определенные в декларативных секциях.
3. Записи из сливаемых файлов копируются в файл-слияния в соответствии с ключами и управляющими фразами ASCENDING или DESCENDING.
4. Файлы закрываются без использования необязательных возможностей, и выполняются все действия, определенные в декларативных секциях.
5. Для варианта GIVING файл с именем имя-файла-3 открывается и в него копируется файл-слияния, после чего этот файл закры-

вается и управление передается оператору, следующему за оператором MERGE.

6. Для варианта OUTPUT PROCEDURE файл-слияния устанавливается в исходное положение и управление передается процедуре вывода, в которой оператор RETURN считывает записи из файла-слияния во внутреннюю память для обработки. После того как в процедуре вывода выполнен завершающий оператор, управление передается оператору, следующему за оператором MERGE.

Глава II. Специальные возможности языка

11.1. Оператор STRING

В языке КОБОЛ величины хранятся во внутренней памяти машины в виде данных. Эти данные именуются с помощью имен-данных и идентификаторов. Идентификатор — это имя-данного, которое может быть уточнено или индексировано для однозначного указания на конкретное данное в памяти. Данное может быть элементарным и групповым. Групповое данное составляется из элементарных данных или других групповых данных. Каждое элементарное данное характеризуется длиной и категорией. Существуют следующие категории элементарных данных: буквенная, буквенно-цифровая, буквенно-цифровая редактируемая, числовая (целая и нецелая) и числовая редактируемая. Фаза USAGE (ДЛЯ) определяет внутреннее представление, используемое для хранения значений данного. Фраза USAGE IS INDEX (ДЛЯ ИНДЕКСА) задает определенное представление адресов памяти машины и идентифицирует данные, не описанные в разделе данных. Фраза USAGE IS COMPUTATIONAL (ДЛЯ ВЫЧИСЛЕНИЙ) (обычно используемая совместно с фразой SYNCHRONIZED (ВЫДЕЛЕНО)) идентифицирует числовую величину, хранящуюся в виде, определенном реализацией и удобном для машинных вычислений. В большинстве случаев это будет двоичное представление числа, но программисту совершенно не обязательно знать его. Фраза USAGE IS DISPLAY (ДЛЯ ВЫВОДА) или отсутствие фразы USAGE вообще означают, что значения должны храниться в виде последовательностей отдельных стандартных литер подобно тому, как они представляются при печати. К числу стандартных литер относятся любые литеры, допустимые на данной вычислительной машине, а не только 71 литера из набора литер КОБОЛа. Фраза USAGE IS DISPLAY может задаваться не только для буквенно-цифровых, но и для числовых данных. При этом допускается выполнение арифметических операций над такими числовыми данными, записанными в виде литер.

Все замечания, касающиеся работы с литерами, сделанные в этом и двух следующих разделах, будут относиться к данным, явно или по умолчанию описанным с помощью фразы USAGE IS DISPLAY.

Обычно литеры, из которых составляется данное, обрабатываются как единое целое, а не по отдельности. Это относится, например, к данным в операторах

MOVE ITEM-A TO ITEM-B.

ADD 123 TO VALUE-C.

Проявив некоторую изобретательность, можно работать с отдельными литерами данного. Например, в описаниях

01 SOME-QUANTITY-X.

05 FULL-DATA-ITEM

PICTURE IS X(15)

USAGE IS DISPLAY.

05 CHARACTER-X

OCCURS 15 TIMES

REDEFINES FULL-DATA-ITEM

PICTURE IS X

USAGE IS DISPLAY.

01 SUBSCRIPT-X

PICTURE IS 9(2)

USAGE IS COMPUTATIONAL

SYNCHRONIZED.

определяется данное FULL-DATA-ITEM, состоящее из пятнадцати литер. Благодаря использованию фразы REDEFINES (ПЕРЕОПРЕДЕЛЯЕТ) и данного SUBSCRIPT-X появляется возможность работы с отдельными литерами из числа этих пятнадцати. Примером может служить оператор:

MOVE CHARACTER-X (SUBSCRIPT-X) TO ANOTHER-PLACE.

При этом необходимо предварительно задать начальное значение данного SUBSCRIPT-X, после чего, выполняя повторно оператор MOVE при различных значениях данного SUBSCRIPT-X, не превосходящих пятнадцати и получающихся, например, с помощью некоторого приращения, можно выделять и обрабатывать подстроки литер данного FULL-DATA-ITEM. Введенные в этих разделах операторы КОБОЛа дают возможность более удобно манипулировать с отдельными символами или строками символов, обеспечивая тем самым гибкость программирования и решение более сложных задач. Эти новые операторы STRING (СОБРАТЬ), UNSTRING (РАЗОБРАТЬ) и INSPECT (ПРОСМОТРЕТЬ) предназначены для работы с одной, несколькими или всеми литерами данного. В любом случае нужно мысленно представлять себе, что литеры различимы и могут перемещаться либо по отдельности, либо, как и раньше, в виде целого данного.

Оператор STRING выбирает строки литер из одного или нескольких пересылаемых данных и, объединяя их в единую строку, помещает в принимающее данное, например:

Пересылаемое данное # 1: ABC

Пересылаемое данное # 2: DEF

Пересылаемое данное # 3: XYZ

Собирая вместе данные 1, 2 и 3, получаем

Принимающее данное: ABCDEFXYZ

Все участвующие в этой сборке данные имеют фиксированную длину, и для принимающего данного этого примера в разделе данных должна была бы быть указана длина, не меньшая девяти позиций литер. Можно было бы написать оператор STRING для сборки данных в любом порядке, для выбора подстрок из пересылаемых данных или для помещения пересылаемых строк в произвольные позиции принимающего данного.

Для этого необходимо ввести понятия ограничителя и указателя. *Ограничитель* — это одна или более литер, временно используемых в качестве специального символа для обозначения конца строки литер. Просмотр литер в КОБОЛе всегда осуществляется слева направо, так что ограничитель всегда должен располагаться на правом конце строки. Если бы в качестве ограничителя использовалась литера «#» (как в приведенном ниже примере), то данное

ABC#DEF#XYZ#

состояло бы из трех подстрок.

Указатель — это идентификатор, который может быть описан как с помощью фразы USAGE IS DISPLAY, так и с помощью фразы USAGE IS COMPUTATIONAL и имеет элементарное целое числовое значение. Это значение представляет собой позицию литеры в некотором другом данном, т. е. оно указывает на определенную позицию литеры в строке. Таким образом, если имеются описания

```
01 POINTER-X      PICTURE IS 999
                   VALUE IS 3.
01 ITEM-X          PICTURE IS X(12)
                   VALUE IS "ABC#DEF#XYZ".
```

то значение данного POINTER-X (УКАЗАТЕЛЬ-X) указывает на третью позицию любого данного. Для данного ITEM-X оно указывает на позицию, занятую литерой «С».

Формальное определение оператора STRING таково:

$$\begin{array}{l} \text{STRING} \left\{ \begin{array}{l} \text{пересылаемый-идентификатор} \\ \text{пересылаемый-литерал} \end{array} \right\} \dots \\ \text{DELIMITED BY} \left\{ \begin{array}{l} \text{идентификатор-1} \\ \text{литерал} \\ \text{SIZE} \end{array} \right\} \dots \\ \text{INTO} \text{ принимающий-идентификатор} \\ \left[\begin{array}{l} \text{WITH POINTER идентификатор-2} \\ \text{ON OVERFLOW повелительные-операторы} \end{array} \right] \end{array}$$

Примеры операторов STRING (смысл операторов STRING будет описан далее) таковы:

STRING ITEM-X DELIMITED BY "BC"

ITEM-Y DELIMITED BY "#"

INTO RECEIVING-ITEM-G WITH POINTER POINTER-X.

STRING "A" DELIMITED BY SIZE

"B" DELIMITED BY SIZE

ITEM-A DELIMITED BY SIZE

INTO ALPHABETIC-ITEM-X

ON OVERFLOW PERFORM ERROR-WRITE-OUT.

Как и для всех операторов КОБОЛа, для этого оператора существуют ограничения:

1. Идентификаторы, за исключением идентификатора-2 фразы POINTER (УКАЗАТЕЛЬ), должны быть описаны в форме для вывода либо с помощью фразы USAGE IS DISPLAY, либо по умолчанию.

2. Принимающий-идентификатор должен обозначать элементарное данное с шаблоном, не содержащим редактирующих символов.

3. Оба литерала могут иметь любую длину, но обязательно должны быть нечисловыми. Они могут быть стандартными константами (например, SPACES), но не должны включать слово ALL. Стандартные константы в операторе STRING (а также в операторах UNSTRING и INSPECT) будут означать только одну литеру упомянутого типа.

Оператор STRING является, по существу, оператором MOVE, в котором содержимое пересылаемого-идентификатора передается в принимающий-идентификатор, как при буквенно-цифровом перемещении. При этом имеет место политерная передача без редактирования. Значения пересылаемых данных никогда не изменяются. В принимающем данном изменяются значения лишь тех позиций литер, в которые помещаются пересылаемые литеры. Например, если бы до сборки значения двух данных были таковы:

ITEM-A	ABC
ITEM-G	GHIJKLM

то после выполнения оператора

STRING ITEM-A DELIMITED BY SIZE INTO ITEM-G.

их значения оказались бы такими:

ITEM-A	ABC
ITEM-G	ABCJKLM

Если фраза **POINTER** не используется, то литеры помещаются в принимающее данное, начиная с крайней левой позиции вправо до тех пор, пока не завершится сборка. Если фраза **POINTER** используется, программист должен обеспечить, чтобы перед выполнением оператора **STRING** указатель имел допустимое значение. Допустимое значение — это любое целое, заключенное между единицей и числом позиций литер принимающего данного включительно. При наличии фразы **POINTER** литеры размещаются, начиная с позиции, определенной значением указателя. После размещения первой литеры значение указателя увеличивается на единицу и передается вторая литера. Это продолжается до тех пор, пока не завершится сборка. Например, если бы выполнялись такие операторы:

```
MOVE 3 TO POINTER-X
STRING ITEM-A DELIMITED BY SIZE
      INTO ITEM-G WITH POINTER POINTER-X.
```

то данные приняли бы значения:

POINTER-X	006
ITEM-A	ABC
ITEM-G	GNABCLM

Значение указателя всегда указывает на следующую заполняемую позицию. Начальное значение указателя не устанавливается оператором **STRING**; этот оператор лишь увеличивает ранее установленное значение. Шаблон указателя должен допускать значения вплоть до числа, на единицу большего, чем размер принимающего данного, с тем, чтобы последнее увеличение указателя не приводило к ошибке. Размещение литер в принимающем данном всегда осуществляется слева направо, меняться может только начальная позиция. Если фраза **POINTER** опущена, то начальной позицией является первая позиция. В противном случае начальная позиция определяется значением указателя.

Литеры, помещаемые в принимающее данное, поступают из пересылаемых данных. Этот процесс всегда начинается с первой литеры пересылаемого данного, записанного в операторе **STRING** первым, и продолжается слева направо. В зависимости от фразы **DELIMITED BY** (**ОГРАНИЧИВАЯСЬ**) передается либо все, либо часть первого данного. После этого передается первая литера второго пересылаемого данного и т. д. до тех пор, пока не завершится сборка. Если в операторе указан вариант **DELIMITED BY SIZE** (**ОГРАНИЧИВАЯСЬ РАЗМЕРОМ**), то пересылаемое данное передается полностью. В противном случае передача отдельного пересылаемого данного оканчивается, когда:

- 1) все литеры будут переданы или
- 2) встретятся литеры, совпадающие со строкой символа ограничителя, определенной либо литералом, либо значением идентификатора-1.

Литера или строка литер, представляющая ограничитель, не передается. Последней передается литера, предшествующая ограничителю, после чего начинается передача литер следующего пересылаемого данного. Если ограничитель в пересылаемом данном не встречается, то это данное передается полностью. Для каждого пересылаемого данного может быть определен лишь один ограничитель; нет возможности завершать передачу по различным ограничителям. Например, для данных

```
01 ITEM-A    PICTURE IS X(10)    USAGE IS DISPLAY
    VALUE IS "AABABCABCD".
01 ITEM-B    PICTURE IS X(7)     USAGE IS DISPLAY
    VALUE IS "XXYXYWZ".
01 RECEIVING-ITEM    PICTURE IS X(12)    USAGE IS
    DISPLAY VALUE IS "IJKLMNOPQRST".
```

в результате выполнения оператора

```
STRING ITEM-A DELIMITED BY "ABC"
      ITEM-B DELIMITED BY "XYZ"
      INTO RECEIVING-ITEM.
```

значение принимающего данного RECEIVING-ITEM станет равным

AABXXYXYWZST

Сборка завершается нормальным образом, когда передана последняя литера последнего пересылаемого данного. Сборка завершается в результате переполнения, либо если делается попытка передать литеру, после того как уже была выполнена передача в последнюю позицию принимающего данного, либо если в начале выполнения оператора STRING указатель имеет недопустимое значение. В случае завершения в результате переполнения при отсутствии варианта OVERFLOW (ПРИ ПЕРЕПОЛНЕНИИ) управление передается оператору, следующему за оператором STRING. Если вариант OVERFLOW указан, то сначала выполняются соответствующие повелительные-операторы.

Заметьте, что в силу ограничения, по которому данные пересылаются слева направо, оператор STRING задает главным образом операцию префиксной конкатенации.

Упражнения

1. Напишите секцию рабочей-памяти и укажите значения данных до и после выполнения параграфа P-2:

P-1.

```
MOVE "1234" TO ITEM-A.  
MOVE "ABCD" TO ITEM-B.  
MOVE "IJKLMNOP" TO ITEM-C.
```

P-2.

```
STRING ITEM-A DELIMITED BY "3"  
SPACES DELIMITED BY SIZE  
ITEM-B DELIMITED BY "CD"  
INTO ITEM-C.
```

2. Пусть даны следующие описания данных:

```
01 ITEM-A    PICTURE IS X(50)  
              USAGE IS DISPLAY.  
  
01 ITEM-B    PICTURE IS X(200)  
              USAGE IS DISPLAY.  
  
01 POINTER-X PICTURE IS 999  
              USAGE IS DISPLAY.
```

Напишите процедуру, которая поместит в данное ITEM-B двадцать пробелов, а затем содержимое данного ITEM-A до комбинации букв «END». Завершите формирование данного ITEM-B присоединением точки, за которой следуют все пробелы.

3. Объедините в одном данном имена, найденные в ста записях входного файла, имеющих такую статью-описания-записи:

```
01 INPUT-RECORD.  
    05 NAME-X    PICTURE IS X(20).  
    05 FILLER    PICTURE IS X(120).
```

где, например, значение данного NAME-X (ИМЯ-X) таково:

JOHN R. SMITH

Отделяйте имена друг от друга запятой и пробелом, а в конце поместите *все пробелы*, например:

JOHN R. SMITH, ROBERT W. DORR, A. B. SEA

Обеспечьте, чтобы за последним именем не следовала запятая.

11.2. Оператор UNSTRING

Назначение оператора STRING заключалось в формировании одного большого данного из нескольких данных или их ведущих частей. Оператор UNSTRING (РАЗОБРАТЬ) служит обратным целям, а именно: разбиению пересылаемого данного на несколько принимающих данных. Например, оператор UNSTRING мог бы разбить данное

ABCDEFXYZ

на несколько других данных

AB
CDEF
XYZ

не меняя исходного данного. Схема разбиения основывается либо на использовании ограничителей, либо на размере принимающего данного. Таким образом, если заданы описания

05 SENDING-ITEM-X PICTURE IS X(9)
 USAGE IS DISPLAY VALUE IS "ABCDEFXYZ".

05 ITEM-A PICTURE IS X(2).

05 ITEM-B PICTURE IS X(4).

05 ITEM-C PICTURE IS X(3).

то выполнение оператора

UNSTRING SENDING-ITEM-X
 INTO ITEM-A ITEM-B ITEM-C

приведет к разбиению строки ABCDEFXYZ на части AB, CDEF и XYZ. Как и раньше, просмотр строки осуществляется слева направо; при этом строка пересылаемого данного расщепляется на подстроки, которые по очереди помещаются в принимающие данные. Литеры каждой подстроки рассматриваются как относящиеся к элементарному буквенно-цифровому данному и помещаются в принимающие данные в соответствии с описаниями их шаблонов. Перемещения, имеющие место при выполнении оператора UNSTRING, обрабатываются как элементарные.

Формальное определение оператора UNSTRING таково:

UNSTRING пересылаемый-идентификатор

$$\left[\begin{array}{l} \text{[DELIMITED BY [ALL] \left. \begin{array}{l} \text{идентификатор-1} \\ \text{литерал-1} \end{array} \right\}} \\ \text{[OR [ALL] \left. \begin{array}{l} \text{идентификатор-2} \\ \text{литерал-2} \end{array} \right\}} \dots \end{array} \right]$$

INTO принимающий-идентификатор

$$\left[\begin{array}{l} \text{[DELIMITER IN идентификатор-3]} \\ \text{[COUNT IN идентификатор-4]} \dots \\ \text{[WITH POINTER идентификатор-5]} \\ \text{[TALLYING IN идентификатор-6]} \\ \text{[ON OVERFLOW побелительные-операторы]} \end{array} \right]$$

Примеры операторов UNSTRING:

UNSTRING ITEM-A DELIMITED BY " , "
INTO REC-A REC-B REC-C
ON OVERFLOW GO TO P-10.

UNSTRING ITEM-A DELIMITED BY "#"
INTO REC-A DELIMITER IN LOCATION-X
REC-B COUNT IN COUNTER-B
REC-C DELIMITER IN LOC-Y COUNT IN A-B.

UNSTRING ITEM-A
INTO REC-A REC-B REC-C REC-D
TALLYING IN COUNT-OF-IDENTIFIERS.

Ограничения на оператор UNSTRING аналогичны ограничениям на оператор STRING.

1. Идентификаторы, за исключением идентификаторов, следующих за словами COUNT (СЧЕТ), POINTER (УКАЗАТЕЛЬ) и TALLYING (СЧИТАЯ), должны быть описаны с использованием для выдачи либо с помощью фразы USAGE IS DISPLAY, либо по умолчанию.

2. Из этих идентификаторов все, за исключением принимающего-идентификатора, должны быть определены как буквенно-цифровые. Ни один из идентификаторов не может быть именем-условия (т. е. данным уровня 88).

3. Идентификаторы пересчета (следующие за словами COUNT, POINTER и TALLYING) должны быть элементарными числовыми

целыми данными и в описании их должна присутствовать фраза **USAGE IS COMPUTATIONAL (ДЛЯ ВЫЧИСЛЕНИЙ)**.

4. Литералы могут быть любой длины, но должны быть нечисловыми. Они могут быть стандартными константами, но при этом могут ссылаться только на одну литеру и не должны включать слово **ALL**.

Перемещение, имеющее место при выполнении оператора **UNSTRING**, начинается либо с самой левой позиции пересылаемого данного, либо с позиции литеры, определенной значением указателя (идентификатора во фразе **POINTER**). Начальное значение этого указателя должно быть установлено до выполнения оператора **UNSTRING**. При выполнении этого оператора указатель получает только приращение. Литеры выделяются из пересылаемого данного и группируются в подстроку до тех пор, пока не произойдет одно из двух:

- 1) длина подстроки оказалась равна длине, определенной шаблоном соответствующего принимающего данного;
- 2) в пересылаемом данном встретилась литера или строка литер, представляющая собой ограничитель.

Эта выделенная подстрока никогда не будет включать сам ограничитель. Выделенная подстрока помещается в подходящее принимающее данное в соответствии с правилами, управляющими элементарным перемещением из буквенно-цифрового данного в принимающее данное, которое может иметь любой шаблон. Ограничитель (если он задан в операторе) передается в идентификатор-3. Если выбор подстроки закончился в результате того, что ее размер стал равен размеру принимающего данного, то в качестве значения идентификатора-3 устанавливаются все пробелы. Фразы **DELIMITER IN (ОГРАНИЧИТЕЛЬ В)** и **COUNT IN (СЧЕТ В)** могут задаваться, только если используется вариант **DELIMITED BY (ОГРАНИЧИВАЯСЬ)**.

Выделение подстроки и элементарное перемещение в принимающее данное продолжают для следующего принимающего данного, указанного в операторе **UNSTRING**, и так далее до тех пор, пока процесс разборки не будет завершен. Разборка завершается нормально, если исчерпаны все литеры пересылаемого данного. Разборка завершается в результате переполнения, если уже использованы все принимающие данные, а в пересылаемом данном еще остались переданные литеры. Переполнение имеет место также в случае, когда перед началом выполнения оператора **UNSTRING** идентификатор-5 из фразы **POINTER** имеет недопустимое значение. Как и в случае оператора **STRING**, при переполнении выполнение оператора **UNSTRING** завершается и, если не задана фраза **ON OVERFLOW (ПРИ ПЕРЕПОЛНЕНИИ)**, выполняется следующий оператор.

Строка ограничителя, состоящая из одной или нескольких литер, описывается фразой DELIMITED BY, в которой могут задаваться несколько ограничителей. Для оператора STRING это не допускалось. Ограничителем может быть любая комбинация литер машины, определенная либо литералом-1, литералом-2 и т. д., либо текущими значениями идентификатора-1, идентификатора-2 и т. д. Непересекающиеся вхождения строки ограничителя в пересылаемое данное представляют собой отдельные вхождения этого ограничителя. Таким образом, если задан оператор

UNSTRING ITEM-A DELIMITED BY "AA"
INTO R-A R-B R-C.

а пересылаемое данное содержит литеры AAA, то ограничитель входит в это данное только один раз. Для наличия в данном двух вхождений ограничителя нужно, чтобы оно содержало, например, литеры AAAA. Продолжая этот пример, отметим, что если бы данное ITEM-A содержало литеры XAAAXAAAA, то значения принимающих данных были бы таковы: X, AX и пробелы. Ниже подчеркнуты отдельные вхождения ограничителя:

XAAAXAAAA

Значением данного R-C были бы либо пробелы, либо нули в зависимости от его фразы PICTURE, так как в пересылаемом данном имеются два расположенных подряд вхождения ограничителя. Так было бы, если бы необязательное ключевое слово ALL (ВСЕМИ) было опущено. При наличии слова ALL все расположенные подряд вхождения ограничителя воспринимаются как одно. Таким образом, если бы в приведенном выше операторе была задана фраза

DELIMITED BY ALL "AA"

то в данном ITEM-A было бы только два вхождения ограничителя

XAAAXAAAA

и значение данного R-C не изменилось бы, оставшись таким, как до выполнения оператора UNSTRING. В этом случае оператор завершается нормально, хотя в данное R-C ничто не помещается.

Указатель (если он задан) увеличивается на единицу для каждой литеры в пересылаемом данном независимо от того, входит ли она в выделенную подстроку или в строку ограничителя. Напомним, что строка ограничителя никогда не передается в принимающее данное; она либо пересылается в идентификатор-3 (полное ее вхождение, например AA или AAAA), либо пропадает, если фраза DELIMITER IN отсутствует. Следовательно, значение указателя будет увеличено на число литер, расположенных, начиная с исходной позиции, вправо до конца пересылаемого данного. Другой

счетчик введен для подсчета числа принимающих данных, фактически участвовавших в конкретном выполнении оператора UNSTRING. Это идентификатор-6 фразы TALLYING IN (СЧИТАЯ В). Точно так же, как и в случае указателя, начальное значение этого счетчика не устанавливается в операторе UNSTRING, а только увеличивается на единицу для каждой выделенной подстроки, пересланной в принимающее данное. В приведенном выше примере, когда менялись данные R-A и R-B, а данное R-C оставалось нетронутым, значение идентификатора-6 было бы увеличено на два.

Последний счетчик представлен идентификатором-4, следующим за словами COUNT IN (СЧЕТ В). Этот счетчик содержит число литер, выделенных в подстроку для соответствующего пересылаемого данного. Это число не включает числа литер строки ограничителя. Начальная установка этого счетчика не нужна, так как подсчитанное значение заносится в идентификатор-4 оператором UNSTRING.

Если во время просмотра пересылаемого данного его последняя литера достигается раньше, чем встретится ограничитель, или раньше, чем длина выделяемой подстроки станет равна размеру принимающего данного, то передается частично сформированная подстрока и осуществляется нормальное завершение оператора. Если литеры строки ограничителя являются последними литерами пересылаемого данного, то в идентификатор-3 фразы DELIMITER IN заносятся все пробелы.

Примеры

1. Пусть заданы следующие статьи-описания-данных:

- | | |
|----------------------------|------------------|
| 01 TALLY-COUNT-X | PICTURE IS 9. |
| 01 POINTER-X | PICTURE IS 99. |
| 01 ITEM-A | PICTURE IS X(8). |
| 01 FIRST-RECEIVING-ITEMS. | |
| 05 R-A | PICTURE IS X(4). |
| 05 D-A | PICTURE IS X(3). |
| 05 C-A | PICTURE IS 99. |
| 01 SECOND-RECEIVING-ITEMS. | |
| 05 R-B | PICTURE IS X(4). |
| 05 D-B | PICTURE IS X(3). |
| 05 C-B | PICTURE IS 99. |
| 01 THIRD-RECEIVING-ITEMS. | |
| 05 R-C | PICTURE IS X(4). |
| 05 D-C | PICTURE IS X(3). |
| 05 C-C | PICTURE IS 99. |

После выполнения операторов

MOVE "ABCDEFGH" TO ITEM-A.

MOVE ZERO TO TALLY-COUNT-X.

UNSTRING ITEM-A INTO R-A R-B TALLYING IN
TALLY-COUNT-X.

новые значения данных будут таковы:

TALLY-COUNT-X	2
R-A	ABCD
R-B	EFGH

Выполнив операторы

MOVE 1 TO POINTER-X.

UNSTRING ITEM-A DELIMITED BY "DEF"

INTO R-A COUNT IN C-A

R-B COUNT IN C-B

WITH POINTER POINTER-X TALLYING IN
TALLY-COUNT-X.

получим следующие новые значения:

POINTER-X	09
TALLY-COUNT-X	4
R-A	ABC
C-A	03
R-B	GH
C-B	02

После выполнения оператора

UNSTRING ITEM-A DELIMITED BY "B" OR "F"

INTO R-A DELIMITER IN D-A COUNT IN C-A

R-B DELIMITER IN D-B COUNT IN C-B

R-C DELIMITER IN D-C COUNT IN C-C.

новые значения будут

R-A	A
D-A	B
C-A	01
R-B	CDE
D-B	F
C-B	03
R-C	GH

D-C

C-C 02

Упражнения

1. Данное, состоящее из одиннадцати литер, содержит номер социального обеспечения, записанный с дефисами, например:

987-65-4320

Напишите оператор UNSTRING и статьи-описания-данных, необходимые для получения единственного девятизначного числа

987654320

2. Данное, состоящее из сорока литер, содержит полное имя человека, причем первой, начиная с левого конца данного, записана фамилия. Фамилия может быть любой длины. Непосредственно за ней следует запятая, а затем имя. Имя также может быть переменной длины и за ним следуют один или более пробелов, а затем инициал. Остаток записи, если он есть, заполнен пробелами. Таким образом, данное

01 FULL-NAME-X PICTURE IS X(40).

могло бы иметь значение:

SMITHSONIAN, ROBERT L

Используя операторы UNSTRING и STRING, напишите процедуру и любые статьи-описания-данных, необходимые для получения сорокалитерного данного, в котором за именем следует инициал с точкой и пробелом, а затем фамилия, например:

ROBERT L. SMITHSONIAN

3. Входная запись содержит текст, в котором за каждым словом следует пробел или точка. Примером такого текста могло бы быть предложение:

NOW IS THE TIME FOR ALL GOOD MEN.

Для анализа текста, например для определения длины слов, частей речи и т. д., желательно разбить текст на отдельные слова, определяя их длину. Напишите процедуру, которая будет разбивать INPUT-RECORD (ВХОДНАЯ-ЗАПИСЬ) на слова (WORD-X) и находить их длины, занося их в данные WORD-LENGTH-X. Для хранения слов и их длин используйте таблицу:

01 WORD-AND-LENGTH-TABLE.

05 INFORMATION-LINE OCCURS 500 TIMES.

10 WORD-X PICTURE IS X(15).

10 WORD-LENGTH-X PICTURE IS 9(2).

4. Используя любые необходимые операторы, напишите полную КОБОЛ-программу для считывания буквенных записей, описанных с помощью фразы PICTURE IS A(80). Каждая запись содержит буквенные слова, отделенные одно от другого одним или более пробелами (например, на правом конце записи может быть шестьдесят пробелов). Уплотните эти записи, переписав их в записи нового файла, имеющие длину около 1000 позиций литер, и заменяя при переписи несколько пробелов одним. Таким образом, три входных записи

```
THE TIME HAS
COME          FOR
ALL GOOD MEN
```

были бы уплотнены в одну запись

```
THE TIME HAS COME FOR ALL GOOD MEN
```

11.3. Оператор INSPECT

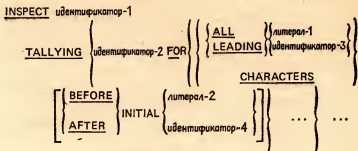
Третьим оператором КОБОЛа, работающим с отдельными литерами внутри данного, является оператор INSPECT (ПРОСМОТРЕТЬ). Он позволяет программисту подсчитывать число вхождений в данное определенных подстрок и дает возможность заменять определенные подстроки другими литерами. При этом подсчет и замена могут быть выполнены с помощью одного оператора INSPECT. Процесс подсчета заключается в просмотре значения данного, например

```
ABCDEFGHNI
```

и в определении числа конкретных литер, например гласных. В данном случае их число равно 003. Замена состоит в замещении выбранных литер какими-либо другими. Например, в рассматриваемом примере можно было бы заменить выбранные гласные на литеру «#»:

```
#BCD#FGH#
```

Подсчитывать или заменять можно как отдельные литеры, так и группы литер, расположенных подряд. Оператор INSPECT, предназначенный для подсчета, имеет формат 1:



Примеры этого оператора:

INSPECT ITEM-A TALLYING SUM-A FOR ALL "A" ALL "E"
ALL "I" ALL "O" ALL "U".

INSPECT ITEM-A TALLYING SUM-A FOR ALL "A"
SUM-B FOR ALL "B"
SUM-C FOR ALL "C".

INSPECT ITEM-A TALLYING SUM-A FOR LEADING
SPACES SUM-B FOR CHARACTERS AFTER
INITIAL "REM".

Идентификатор-1, соответствующий просматриваемому данному, может именовать групповое данное или элементарное данное любой категории с использованием для выдачи, определенным либо фразой USAGE IS DISPLAY, либо по умолчанию. Идентификатор-2 может именовать только элементарное числовое данное, которое может быть описано с использованием для вычислений (USAGE IS COMPUTATIONAL). Оставшиеся два идентификатора: идентификатор-3 и идентификатор-4 не должны принадлежать к редактируемой категории и должны быть описаны с использованием для выдачи (явно или по умолчанию). Они могут быть числовыми, буквенными, а также буквенно-цифровыми, но должны быть элементарными. Литералы должны быть нечисловыми. Если они являются стандартными константами, то должны ссылаться только на одну литеру и не могут включать слово ALL.

Так же как и в операторах STRING и UNSTRING, просмотр литер осуществляется слева направо. В рассматриваемом операторе отсутствует указатель (фраза POINTER), так как при каждом его выполнении просматриваются все литеры данного с идентификатором-1, начиная с первой позиции. При просмотре литеры всегда трактуются как буквенно-цифровые, даже если идентификатор-1 имеет числовой шаблон. Кроме того, если идентификатор-1 является числом со знаком, то в операторе INSPECT знак полностью игнорируется. Начальное значение данного с идентификатором-2, являющего-

с счетчиком, не устанавливается в операторе INSPECT. Его значение увеличивается на единицу для каждого непересекающегося вхождения набора литер, указанного после ключевого слова FOR. Следующая после слова FOR фраза

$$\left\{ \begin{array}{l} \text{ALL} \\ \text{LEADING} \end{array} \right\} \left\{ \begin{array}{l} \text{литерал-1} \\ \text{идентификатор-3} \end{array} \right\} \left[\begin{array}{l} \text{BEFORE} \\ \text{AFTER} \end{array} \right] \text{INITIAL} \left\{ \begin{array}{l} \text{литерал-2} \\ \text{идентификатор-4} \end{array} \right\} \\ \text{CHARACTERS}$$

может повторяться желаемое число раз. Если указано слово ALL (ВСЕ), то подсчитываются все непересекающиеся вхождения значения либо литерала-1, либо идентификатора-3 в соответствии с уточнением, задаваемым необязательной фразой BEFORE/AFTER (ДО/ПОСЛЕ). Например, при наличии в операторе INSPECT фразы

ALL "AA" AFTER INITIAL "C"

для значения идентификатора-1

AABAACAAAAXAAA

было бы обнаружено три вхождения строки «AA».

Если задано слово LEADING (ВЕДУЩИЕ), то подсчитываются только крайние слева расположенные подряд (не прерываемые другими литерами) вхождения. Например, для приведенного выше значения идентификатора-1 при наличии фразы

LEADING "AA"

было бы обнаружено одно вхождение. Комбинация слов LEADING и BEFORE смысла не имеет, но если бы была задана фраза

LEADING "AA" AFTER INITIAL "C"

то подсчет вхождений начинался бы только после обнаружения литеры «C». В приведенном выше примере для данной фразы было бы обнаружено два вхождения строки «AA».

Если задано слово CHARACTERS (ЛИТЕРЫ), то следует использовать необязательную фразу BEFORE/AFTER, в противном случае увеличение счетчика, заданного идентификатором-2, будет просто равно размеру идентификатора-1. При том же самом значении идентификатора-1 для фразы

CHARACTERS AFTER INITIAL "C"

значение счетчика увеличится на восемь. При использовании фразы BEFORE/AFTER операция просмотра строки либо завершается непосредственно перед подстрокой, указанной после слова INITIAL, либо начинается сразу же после этой подстроки. Ни в том, ни в другом случае сама эта подстрока при подсчете вхождений не учиты-

вается. В любом случае не забывайте устанавливать начальное значение идентификатора-2, так как в операторе INSPECT оно не устанавливается, а только увеличивается в процессе подсчета.

Второй процесс, обеспечиваемый оператором INSPECT, — это процесс замещения. Для него введен специальный формат 2:

INSPECT идентификатор-1

$$\begin{array}{c} \text{REPLACING} \left\{ \left\{ \begin{array}{l} \text{ALL} \\ \text{LEADING} \\ \text{FIRST} \end{array} \right\} \left\{ \begin{array}{l} \text{литерал-3} \\ \text{идентификатор-5} \end{array} \right\} \text{BY} \left\{ \begin{array}{l} \text{литерал-4} \\ \text{идентификатор-6} \end{array} \right\} \right. \\ \left. \left\{ \begin{array}{l} \text{BEFORE} \\ \text{AFTER} \end{array} \right\} \text{INITIAL} \left\{ \begin{array}{l} \text{литерал-5} \\ \text{идентификатор-7} \end{array} \right\} \dots \right\} \dots \end{array}$$

Примеры:

INSPECT ITEM-A REPLACING ALL "A" BY "#"
"E" BY "#".

INSPECT ITEM-A REPLACING LEADING SPACES BY "*"
ALL ZEROES BY SPACES AFTER INITIAL ".".

INSPECT ITEM-A REPLACING FIRST "*" BY "\$".

В этих примерах стандартные константы SPACES (ПРОБЕЛЫ) и ZEROES (НУЛИ) обозначают только одну соответствующую литеру. Заменяющая подстрока литер (из одной или более литер) должна быть того же размера, что и заменяемая подстрока. Поэтому следующее не допускается:

INSPECT A-A REPLACING ALL "\$" BY "ABC". (НЕВЕРНО!)

При замене, так же как и при подсчете, просмотр строки осуществляется слева направо, начиная с первой позиции и до конца строки. Каждый раз, когда встречается комбинация литер, совпадающая с любой из комбинаций, заданных во фразе REPLACING (ЗАМЕНЯЯ), делается замена. Если в какой-либо позиции идентификатора-1 уже была произведена замена литеры, дальнейших замен литеры в этой позиции не производится и новая литера в последующих сравнениях не участвует. Ключевые слова ALL (ВСЕ), LEADING (ВЕДУЩИЕ) и FIRST (ПЕРВЫЙ) имеют свой обычный смысл и применяются ко всем повторяющимся фразам, которые следуют за ними в операторе INSPECT. Слово ALL используется для замены всех непересекающихся вхождений заменяемых подстрок в идентификатор-1 с учетом уточняющей фразы BEFORE/AFTER. Слово LEADING используется для замены всех самых левых расположенных подряд вхождений заменяемой подстроки, при этом самое левое вхождение может быть уточнено с помощью фразы AFTER INITIAL. Слово FIRST используется для замены только первого

вхождения заменяемой подстроки, считая слева направо с учетом фразы BEFORE/AFTER; в данном случае слово BEFORE имеет смысл, так как ограничивает справа поиск нужной подстроки.

Существует множество возможных применений варианта REPLACING. Его можно использовать для подавления ведущих нулей:

INSPECT ITEM-G REPLACING LEADING ZEROES BY SPACES.

Но этого же, конечно, можно добиться посредством оператора MOVE, описывая принимающее поле с помощью фразы PICTURE, использующей управляющие символы подавления нулей — Z. Однако оператор INSPECT может быть использован и для обратного, т. е. для замены пробелов нулями:

INSPECT ITEM-G REPLACING LEADING SPACES BY ZEROES.

Оператор INSPECT можно использовать также для составления отчетов, где часто желательно выделять отрицательные итоги, или убытки, заключая их в круглые скобки, а не используя символы DB (ДБ) и CR (КР). Для получения плавающей левой круглой скобки можно использовать шаблон вида

PICTURE IS ————.99

При помещении значения в такое числовое редактируемое данное вместо последнего ведущего нуля будет вставлена литера «—», а в предшествующие ей позиции будут вставлены пробелы. После этого остается применить оператор.

INSPECT ITEM-A REPLACING FIRST "—" BY "(".

Положение правой круглой скобки фиксировано, и она могла бы быть добавлена с помощью элементарного данного с фразой VALUE IS ")", следующего непосредственно за данным ITEM-A.

С помощью оператора INSPECT можно проводить анализ текста. Он также может помочь в расшифровке криптограмм. Например, считывая буквенную запись и применяя последовательно операторы вида

INSPECT TEXT-RECORD TALLYING SUM-A FOR ALL "E"
SUM-B FOR ALL "T" SUM-C FOR ALL "A".

можно подсчитать частоты вхождений различных букв.

Кроме того, с помощью этого оператора можно выполнять перекодировку колод перфокарт. На устройствах перфорации различных типов все еще используются для кодировки специальных литер различающиеся комбинации отверстий в колонках пер-

фокарт. На КОБОЛе можно написать программу, которая считывала бы карты и после перекодировки с помощью, например, такого оператора

```
INSPECT IN-RECORD REPLACING ALL "$" BY "="
      "€" BY "+" "?" BY QUOTE.
```

перфорировала бы новые перфокарты.

Оба варианта TALLYING (СЧИТАЯ) и REPLACING (ЗАМЕНЯЯ) можно объединить в одном операторе, записывая сначала фразу TALLYING, для получения составной операции над идентификатором-1. Все подсчеты осуществляются раньше любой замены. Например:

```
INSPECT ITEM-A TALLYING L-COUNT FOR ALL "L"
      REPLACING ALL "E" BY "I" AFTER INITIAL "L".
INSPECT ITEM-A TALLYING CHARACTER-COUNT FOR
      CHARACTERS AFTER INITIAL ":" REPLACING
      LEADING "QU" AFTER INITIAL ":".
```

Упражнения

Пусть даны следующие статьи-описания-данных:

01 LIST-OF-ITEMS.

05 SINGLE-ITEMS

OCCURS 100 TIMES

PICTURE IS X

USAGE IS DISPLAY.

01 SUBSCRIPT-X

PICTURE IS 999

USAGE IS COMPUTATIONAL
SYNCHRONIZED.

Напишите процедуру для помещения первой отличной от пробела литеры данного LIST-OF-ITEMS в данное с именем FIRST-NON-BLANK-CHARACTER.

2. Напишите процедуру и статьи-описания-записей для считывания исходной КОБОЛ-программы и замены каждого вхождения слова RUN на слово A-X. Будьте внимательны, чтобы не заменить слово RUNNING на A-XNING или RUNS на A-XS и т. д.

3. Напишите полную программу для чтения нескольких (не более десяти) входных записей, каждая из которых состоит из восьми-десяти буквенных литер. Распечатайте частоты вхождения различных букв, иными словами, число появлений буквы A, число появлений буквы B и т. д.

4. Напишите процедуру проверки и (если нужно) преобразования входного поля записи, состоящего из двадцати пяти литер. Это поле должно быть буквенным и содержать буквы в первой пози-

ции. Между буквами не должно быть пробелов, хотя правее буквенного слова, содержащегося во входном поле, может быть любое число пробелов. Это слово должно состоять не менее чем из десяти букв.

11.4. Сегментация

В большинстве вычислительных машин объем внутренней памяти, доступной для использования программой, ограничен. Язык КОБОЛ обеспечивает различные возможности уменьшения объема памяти, выделяемой под данные. Это достигается путем:

1) употребления либо фразы SAME AREA (ОБЩАЯ ОБЛАСТЬ), в результате чего для различных файлов, если только они не открыты одновременно, используется одна и та же область памяти, либо фразы SAME RECORD AREA (ОБЩАЯ ОБЛАСТЬ ЗАПИСИ), в результате чего несколько файлов, которые могут быть открыты одновременно, используют общую область памяти для обработки текущей логической записи;

2) применения фразы REDEFINES (ПЕРЕОПРЕДЕЛЯЕТ) к одной и той же области памяти для двух различных данных опять же при условии, что оба значения не существуют одновременно.

Общий прием, используемый в обоих случаях, называется «перекрытием», так как различные данные перекрываются или занимают в разные моменты времени одно и то же место в памяти. Тот же самый подход, который применяется для буферов и записей файлов и для данных во внутренней памяти, может быть с равным успехом применен к области памяти, в которой хранится рабочая программа. Обычно не требуется, чтобы вся программа одновременно находилась в памяти. Как правило, имеет место естественная или удобная сегментация программы на отдельные участки, соответствующие различным фазам выполнения, так что в любой момент времени в память может быть загружен лишь один из таких участков.

Уже отмечалось, что параграфы не обязательно должны объединяться в секции, но все же чаще программы составляются из секций. Использование секций обязательно для реализации перекрытий в программе. Если программа написана без объединения параграфов в секции, то для того, чтобы перекрытия стали возможны, необходимо такое объединение провести. Несколько секций могут быть объединены в сегмент. Принадлежность секции к сегменту указывается с помощью номера-сегмента, записываемого после заголовка-секции, например:

FIND-MAXIMUM SECTION 20.

Номер-сегмента должен быть целым от 0 до 99 включительно. Если номер-сегмента опущен, то предполагается, что он равен нулю.

Следовательно, все приводившиеся до сих пор секции относились к сегменту 0. Формальное определение таково:

имя-секции SECTION номер-сегмента.

Сегмент, состоящий из любого числа секций, загружается во внутреннюю память как единый блок рабочей программы. За исключением отмеченного ниже случая, все сегменты с номерами, меньшими 50 (от 0 до 49 включительно), загружаются в память как один непрерывный блок и остаются в памяти постоянно. Все сегменты с номерами от 50 и выше помещаются во внутреннюю память, только когда во время работы программы к ним произошло обращение, например при помощи оператора GO TO. Каждый раз, когда новый сегмент с номером 50 или выше помещается в память, он перекрывает любой ранее размещенный в памяти сегмент с номером, большим или равным 50. Сегменты с номерами, меньшими 50 (за одним исключением, о котором будет сказано позже), не перекрываются и остаются в памяти постоянно. Такое перекрытие позволяет использовать большие программы на машинах с малой памятью при условии, что некоторые из программ могут быть разбиты на сегменты так, что в любой момент во внутренней памяти должна храниться лишь часть из этих сегментов. На самом деле все программы могут быть сегментированы подобным образом, так как в память может помещаться лишь текущий программный сегмент. Однако при этом программа будет работать намного дольше, потому что потребуется определенное время на загрузку этих сегментов из вспомогательной памяти.

Сегменты с номерами, большими или равными 50, называются независимыми сегментами. Они хранятся и перекрываются в области памяти, отдельной от той, в которой хранятся сегменты с номерами, меньшими 50. Эти последние сегменты называются постоянными сегментами. Они загружаются в память один раз и остаются там до конца работы программы. Единственная причина, по которой для постоянных сегментов выделяется больше одного номера, заключается в обеспечении совместимости между разными вычислительными машинами. Если программа должна работать на машине с объемом памяти, меньшим, чем предусмотрено для постоянных сегментов, то, используя необязательную фразу в параграфе OBJECT-COMPUTER (РАБОЧАЯ-МАШИНА), можно изменить границу между постоянными и независимыми сегментами. Эта новая фраза такова:

OBJECT-COMPUTER. имя-машины

SEGMENT-LIMIT IS номер-сегмента.

Например:

OBJECT-COMPUTER.

B-6700

SEGMENT-LIMIT IS 30.

Все постоянные сегменты с номерами от 30 до 49 включительно получили бы при этом статус, аналогичный статусу независимых сегментов. Такие сегменты называются фиксированными перекрываемыми сегментами. Они могут быть перекрыты сами или могут перекрывать другие сегменты. Единственное различие между фиксированными перекрываемыми сегментами и независимыми сегментами заключается в том, что фиксированный перекрываемый сегмент каждый раз загружается в память в том состоянии, в котором он находился перед тем, как его в последний раз перекрыли, а независимый сегмент всегда загружается в память в исходном состоянии. (Однако это различие незначительно, и в большинстве случаев им можно пренебречь.)

На секции, записанные в декларативной (DECLARATIVES) части раздела процедур, всегда накладываются особые ограничения. Они не должны содержать номеров-сегментов, больших 49, и всегда являются частью постоянного сегмента.

Повторим, что программа должна быть разбита программистом не две главные части: фиксированную и постоянную часть, которая остается в памяти, и набор загружаемых сегментов, которые размещаются в памяти по мере необходимости. Пример использования номеров-сегментов:

PROCEDURE DIVISION.

FIRST-PERMANENT SECTION 10.

P-1. GO TO G-1.

P-2. DISPLAY "Y".

SECOND-INDEPENDENT SECTION 50.

H-1. DISPLAY "Z".

STOP RUN.

THIRD-INDEPENDENT SECTION 60.

G-1. DISPLAY "X"

GO TO P-2.

Хотя в примере указаны три сегмента, в любой момент времени в памяти будут находиться только два из них — это постоянный сегмент и один из независимых сегментов. В данном примере каждый сегмент состоит только из одной секции, но это сделано лишь для краткости. На самом деле в любом сегменте может быть столько секций, сколько необходимо программисту, при условии что объем

памяти достаточен для хранения всего сегмента. Такие секции не обязательно должны быть расположены в программе подряд, и нумерация сегментов не влияет на логический поток команд. В приведенном выше примере сначала загружается секция **FIRST-PERMANENT** (ПЕРВАЯ-ПОСТОЯННАЯ) и выполнение начинается с параграфа P-1. Оператор **GO TO** передает управление параграфу G-1, что приводит к загрузке всех секций (в данном случае одной) сегмента 60 в область независимых сегментов. Выдается значение «X», и управление передается параграфу P-2, в котором выдается значение «Y». Затем в результате нормальной последовательной передачи управления управление передается параграфу H-1. Это приводит к вызову сегмента 50, который загружается в область, занятую до этого сегментом 60. Выдается значение «Z», и работа завершается.

Существуют незначительные ограничения на операторы **PERFORM** (ВЫПОЛНИТЬ), имеющие дело с перекрывающимися сегментами. Оператор **PERFORM**, находящийся в сегменте с номером, меньшим, чем определенная граница-сегментов (**SEGMENT-LIMIT**) (т. е. в постоянной секции), может обращаться только к секциям с номерами-сегментов, меньшими 50, или к секциям, которые все содержатся в одном сегменте, если номер-сегмента больше или равен 50. Оператор **PERFORM**, расположенный в секции с номером, равным или большим, чем граница-сегментов (т. е. в перекрываемой секции), может обращаться только к секциям, имеющим тот же самый номер-сегмента, или к секциям с номером-сегмента, меньшим, чем граница-сегментов.

11.5. Средства коммуникации

До сих пор описывались не прямые способы ввода данных в машину, а именно способы, при которых исходные значения данных записываются на носитель памяти, такой, как перфокарты или магнитная лента, а затем файл карт или катушка магнитной ленты читается во внутреннюю память. В таком не прямом вводе данных существуют определенные преимущества. Во-первых, запись информации на носитель делается независимо от вычислительной машины, что освобождает ее для другой работы. Возможные ошибки в исходных данных могут быть обнаружены и исправлены без использования обрабатывающей эти данные программы. Во-вторых, входные данные могут быть сгруппированы. Иными словами, можно накапливать данные, поступающие за некоторый период времени до тех пор, пока их не накопится достаточно для того, чтобы их обработка на машине могла быть сделана эффективно. В не прямой передаче данных имеются также и недостатки. Как правило, данные подготавливаются для ввода людьми, не знакомыми с этими данными, и

тем самым в систему вносятся «очевидные» ошибки. Отсрочка внесения новых данных, вызванная их группированием, может привести к ошибкам в системах административного управления: бухгалтерские записи, сделанные в начале месяца, не будут отражены в отчетах до подведения итогов в конце месяца. Группировать данные в системах административного управления — это то же самое, что использовать личную чековую книжку, не делая никаких отметок о суммах, выплаченных по чекам. Если на счете нет достаточно большой суммы, то часто выплата по выписанным чекам не сможет быть произведена.

На многих вычислительных установках возможен прямой ввод данных. Он позволяет пользователю самостоятельно вводить данные и получать ответ непосредственно и быстро. Такой ввод осуществляется с помощью группы устройств, именуемых *терминалами*. Терминалы могут быть телетайпами, клавиатурами, подсоединенными к машине, или даже двенадцатью кнопками на телефоне. Обычно эти терминалы расположены на значительном расстоянии от машины и нуждаются в коммуникационной сети для передачи данных. Коммуникационная сеть может быть выполнена на базе радиоволн микроволнового диапазона, инфракрасного излучения или телефонного кабеля. До сих пор наиболее распространенным методом передачи данных на расстояние остается использование телефонных линий, будь то обычные телефонные линии или специально предназначенные для этого кабели. В силу такой важности коммуникационной сети для прямого ввода данных средства КОБОЛа, с помощью которых обрабатываются такие данные, называются средствами коммуникации.

Терминалы обычно бывают двусторонними: они могут передавать в машину запись данных (называемую в данном контексте сообщением) или принимать такую запись из машины. Для передачи сообщения в машину используются телетайпоподобные клавишные пульты, а для получения ответных сообщений из машины применяются небольшие печатающие устройства или видеодисплеи. Представьте себе телетайп на конце длинного кабеля, соединенного с вычислительной машиной, и вы получите хороший образец терминала. Вы можете печатать сообщения для вычислительной машины и она может печатать ответные сообщения. Конечно, на многих специализированных установках, таких, как банковские установки, установки бюро предварительных заказов авиационных билетов или установки больших универмагов, будут использоваться специально разработанные терминалы. Но для каждого из таких терминалов будут иметь место передача и прием сообщений.

Между внутренней памятью и терминалами находятся буферы, именуемые *очередями сообщений* (точно так же, как существуют области буферной памяти, связывающие внутреннюю память с внешними файлами на лентах и дисках). При передаче сообщения с неко-

торого терминала оно устанавливается в очередь и ждет в этой очереди до тех пор, пока КОБОЛ-программа не вызовет его во внутреннюю память. Если терминалы передают сообщения быстрее, чем КОБОЛ-программа может их обработать, то они выстраиваются в очередь, ожидая обработки, подобно людям, выстраивающимся в очередь к билетной кассе. Для передачи сообщений из машины на терминалы существуют выходные очереди. Поэтому, когда КОБОЛ-программа посылает сообщение на один из терминалов, это сообщение сначала устанавливается в соответствующую очередь и ждет готовности терминала. Входная очередь может быть своя для каждого терминала или одна для всех терминалов. Очереди могут даже состоять из подочереди. Сложность организации очередей зависит от числа и типа терминалов, частоты передач и длины передаваемых сообщений, от желаемого времени ответа.

В записи описания-коммуникации (communication-description record), хранящейся во внутренней памяти и первоначально устанавливаемой КОБОЛ-программой, указываются источник сообщения или адресат, и другая управляющая сообщениями информация. Эта запись CD (OK) управляет частью операционной системы, именуемой *системой управления сообщениями* (Message Control System—MCS), и в свою очередь получает из MCS определенную информацию, такую, как дата и время появления сообщения, и число литер, содержащихся в сообщении. Схема, демонстрирующая взаимосвязь между терминалами, линиями коммуникации, очередями, MCS и внутренней памятью, приведена на рис. 11.1 Каждому терминалу, каждой очереди, всем записям CD и, конечно, внутренним данным присвоены уникальные имена: терминал «A43», очередь «ZIP» и т. д. Само сообщение имени не имеет. Имена имеют только величины, влияющие на сообщение. Эти имена указываются системе управления сообщениями с помощью языка управления операционной системой, и программист должен знать, что это за имена, перед тем, как приступить к программированию.

Для передачи сообщений между очередями сообщений и КОБОЛ-программой в КОБОЛе используются два оператора. Это операторы SEND (ПОСЛАТЬ) и RECEIVE (ПОЛУЧИТЬ). Оператор SEND заносит сообщение в одну из выходных очередей, откуда впоследствии оно будет передано с помощью MCS определенному принимающему терминалу. Оператор RECEIVE перемещает очередное доступное сообщение (выбираемое с соответствии с правилом: первый пришел—первый обслужен) из поименованной входной очереди во внутреннее данное. Для этого сообщение было занесено в эту входную очередь с помощью MCS. Программист должен знать, какой терминал к какой очереди приписан. Передача сообщения операторами SEND и RECEIVE осуществляется под управлением соответствующей записи описания-коммуникации (задаваемой в статье CD). Записи CD описываются в специальной секции раздела данных. Это секция

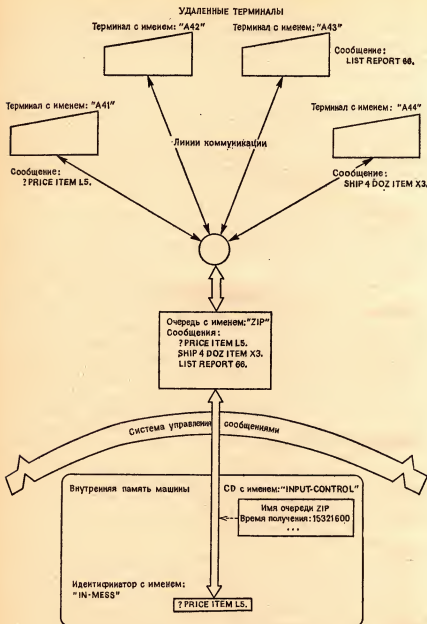


Рис. 11.1. Схема средств коммуникации в КОБОЛе.

коммуникаций (COMMUNICATION SECTION). В разделе данных имеются следующие секции, которые должны записываться в указанном порядке:

FILE SECTION.

WORKING-STORAGE SECTION.

LINKAGE SECTION.

COMMUNICATION SECTION.

Секция коммуникаций содержит статьи CD. Для статей CD существуют два формата: один для приема входного (INPUT) сообщения (используемый оператором RECEIVE), другой для передачи выходного (OUTPUT) сообщения (используемый оператором SEND). Каждая статья CD порождает запись, которая будет содержать значения, связанные с передачей конкретного сообщения. Входная статья CD имеет следующий общий формат:

CD имя-ок FOR INPUT

SYMBOLIC QUEUE IS имя-данного-1

[MESSAGE DATE IS имя-данного-2]

[MESSAGE TIME IS имя-данного-3]

[SYMBOLIC SOURCE IS имя-данного-4]

[TEXT LENGTH IS имя-данного-5]

Выходная статья CD имеет другой формат:

CD имя-ок FOR OUTPUT

SYMBOLIC DESTINATION IS имя-данного-1

[TEXT LENGTH IS имя-данного-2]

Примеры статей CD:

DATA DIVISION.

. . .

COMMUNICATION SECTION.

CD INPUT-CONTROL FOR INPUT

SYMBOLIC QUEUE IS PLACE-HOLDING-QUEUE-NAME

MESSAGE TIME IS PLACE-TIME-WILL-BE-STORED.

CD OUTPUT-CONTROL FOR OUTPUT

SYMBOLIC DESTINATION IS ITEM-CONTAINING-

TERMINAL-NAME.

Обратите внимание, что значениями имен-данных являются либо имена-очередей, либо точное время сообщения. Программист дол-

жен задавать имена исходной очереди (QUEUE) или принимающего терминала (DESTINATION) в виде буквенно-цифровых литералов (в данном случае PLACE-HOLDING-QUEUE-NAME и ITEM-CONTAINING-TERMINAL-NAME). После того как заданы описания, могут быть выполнены операторы RECEIVE и SEND. Формат оператора RECEIVE таков:

RECEIVE имя-ок MESSAGE INTO идентификатор
[NO DATA повелительные-операторы]

где имя-ок должно быть именем входной статьи CD (OK). Очередное сообщение из очереди, определенной значением данного с именем имя-данного-1, будет помещено в данное, определенное идентификатором. Например:

MOVE "ZIP" TO PLACE-HOLDING-QUEUE-NAME.
 RECEIVE INPUT-CONTROL MESSAGE
 INTO MESSAGE-RECORD
 NO DATA GO TO P-10.

Если в поименованной очереди больше нет сообщений, то будут выполняться повелительные-операторы фразы NO DATA (НЕТ ДАННЫХ). Если фраза NO DATA опущена, программа прервется на операторе RECEIVE до тех пор, пока в очереди не появится какое-либо сообщение. В зависимости от логики программы подходящим может оказаться либо то, либо другое.

Формат оператора SEND таков:

SEND имя-ок FROM идентификатор

Например:

MOVE "A42" TO ITEM-CONTAINING-TERMINAL-NAME.
 SEND OUTPUT-CONTROL FROM LOCATION-IN-MEMORY.

Обратите внимание, что адресатом сообщения в операторе SEND является имя терминала, в то время как источником сообщения для оператора RECEIVE является имя-очереди. Для коммуникационных передач данных операторы OPEN или CLOSE не используются. Коммуникационная сеть активируется, когда начинает работу система управления сообщениями; это делается без участия КОБОЛ-программы. Если коммуникационная линия должна быть разорвана или терминал следует отключить, MCS известит об этом оператора вычислительной машины. Кобол-программа имеет дело лишь с входными и выходными очередями, оставляя все остальные функции передачи данных на долю MCS.

Пример программы, использующей средства связи, приведен ниже. Предполагается, что программа получает с терминала банковского кассира сообщения вида

+15-6753 00568950

где первый символ означает вклад в банк, а не изъятие, второе слово — это номер счета клиента, а последнее слово означает внесенную сумму, в которой в двух последних позициях указываются центы. КОБОЛ-программа получает сообщение, затем с помощью оператора UNSTRING разбирает его на три описанные выше части и обновляет запись, хранящуюся на устройстве массовой памяти. Эта программа является примером, и поэтому в нее не встроены процедуры проверки, гарантирующие правильность сделанных изменений. В реальной ситуации прежде, чем фактически выполнить обновление записи, программа отослала бы предполагаемые изменения кассиру для их визуальной проверки. Но, конечно, цель примера — проиллюстрировать передачу сообщений, а не написать полную реальную программу.

FILE-CONTROL.

```
SELECT INDEX-MASS-STORAGE-FILE
  ASSIGN TO DISK
  ORGANIZATION IS INDEXED
  ACCESS MODE IS RANDOM
  RECORD KEY IS ACCOUNT-NUMBER.
```

. . .

DATA DIVISION.

FILE SECTION.

FD INDEX-MASS-STORAGE-FILE

LABEL RECORDS ARE STANDARD.

01 INDEXED-RECORD.

05 ACCOUNT-NUMBER	PICTURE IS X(7).
05 PERSONS-NAME	PICTURE IS X(25).
05 LATEST-BALANCE	PICTURE IS 9(6)V99.
05 FILLER	PICTURE IS X(300).

WORKING-STORAGE SECTION.

77 DEPOSIT-OR-WITHDRAWAL-CODE	PICTURE IS X.	
77 L-VALUE	PICTURE IS 99	VALUE IS 28.
77 TRANSACTION-AMOUNT	PICTURE IS 9(6)V99.	

01 INPUT-MESSAGE PICTURE IS X(87).
 01 OUTPUT-MESSAGE.
 05 ACCOUNT-NUMBER PICTURE IS X(7).
 05 FILLER PICTURE IS X(10) VALUE IS SPACES.
 05 LATEST-BALANCE PICTURE IS \$(4),\$(3).99.

COMMUNICATION SECTION.

CD INPUT-CD-CONTROL FOR INPUT
 SYMBOLIC QUEUE IS QUEUE-NAME
 SYMBOLIC SOURCE IS TERMINAL-NAME.

CD OUTPUT-CD-CONTROL FOR OUTPUT
 SYMBOLIC DESTINATION IS
 DESTINATION-NAME
 TEXT LENGTH IS L-VALUE.

PROCEDURE DIVISION.

P-1.

OPEN I-O INDEX-MASS-STORAGE-FILE.
 MOVE "ZIP" TO QUEUE-NAME.

P-2.

RECEIVE INPUT-CD-CONTROL MESSAGE INTO
 INPUT-MESSAGE.

- * ОТСУТСТВИЕ ФРАЗЫ NO DATA ОЗНАЧАЕТ, ЧТО
- * ПРОГРАММА БУДЕТ ЖДАТЬ В ЭТОМ МЕСТЕ ДО ТЕХ ПОР,
- * ПОКА ПО КРАЙНЕЙ МЕРЕ ОДНО СООБЩЕНИЕ НЕ
- * ПОСТУПИТ В ОЧЕРЕДЬ INPUT-QUEUE.

UNSTRING INPUT-MESSAGE INTO DEPOSIT-OR-
 WITHDRAWAL-CODE ACCOUNT-NUMBER IN
 INDEXED-RECORD TRANSACTION-AMOUNT.
 READ INDEX-MASS-STORAGE-FILE RECORD
 INVALID KEY STOP RUN.

IF DEPOSIT-OR-WITHDRAWAL-CODE IS EQUAL TO "+"
 ADD TRANSACTION-AMOUNT TO LATEST-
 BALANCE IN INDEXED-RECORD
 ELSE SUBTRACT TRANSACTION-AMOUNT FROM
 LATEST-BALANCE IN INDEXED-RECORD.

REWRITE INDEXED-RECORD INVALID KEY
STOP RUN.

MOVE CORRESPONDING INDEXED-RECORD TO
OUTPUT-MESSAGE.

MOVE TERMINAL-NAME TO DESTINATION-NAME.

- * ЭТОТ ОПЕРАТОР MOVE ПОМЕЩАЕТ ИМЯ ТЕРМИНАЛА
- * В ВЫХОДНУЮ ЗАПИСЬ CD.

SEND OUTPUT-CD-CONTROL FROM OUTPUT-MESSAGE.
GO TO P-2.

11.6. Средства отладки

Программу для вычислительной машины нельзя назвать программой в полном смысле этого слова до тех пор, пока она не начнет работать, иными словами, пока она не будет преобразовывать входные данные в выходные записи в соответствии с алгоритмом решаемой задачи. Достигнуть этой цели, как правило, бывает трудно для всякой программы независимо от ее размера. Начинающего программиста обычно беспокоит синтаксис статей и операторов КОБОЛа. Опытные программисты на самом деле часто довольно небрежно относятся к синтаксису, или грамматике, программы по двум причинам: во-первых, почти в любом компиляторе имеются мощные диагностические средства, обнаруживающие и распечатывающие почти каждую грамматическую ошибку, и, во-вторых, гораздо более опасны семантические ошибки или запутанный смысл программы. Например, такой оператор

AID ITEM-A TO ITEM-B

не будет принят из-за ошибки в написании арифметического глагола, но оператор

ADD ITEM-A TO ITEM-B

будет работать и порождать результаты, которые будут неверны, если в программе необходимо перемножать эти два данных, а не складывать их.

Число семантических или логических ошибок в программе может быть уменьшено благодаря наличию точных и детализированных исходных спецификаций алгоритма решения задачи с помощью использования блок-схем и комментариев, документирующих каждый

шаг решения, с помощью подготовки тестовых данных для проверки работы программы и с помощью контроля фактического исполнения программы. Средства отладки КОБОЛа предназначены для содействия именно в таком контроле и для обеспечения возможностей как слежения за логическими шагами работающей программы, так и вывода значений определенных данных в процессе вычислений.

Средства для осуществления отладки, предоставляемые КОБОЛом программисту, реализуются в виде строк отладки и секций отладки. Строки отладки могут быть любыми статьями или операторами КОБОЛа, добавляемыми в программу для наблюдения за промежуточными значениями и операциями во время выполнения программы. Такие строки будут исключены из процесса выполнения программы, после того, как программа будет отлажена и готова для использования. При этом не обязательно фактически удалять их из программы. Они просто не будут компилироваться при окончательной компиляции готовой программы. Секции отладки — это целые секции, включаемые в секции декларатив разделов процедур. Строки отладки выполняются каждый раз, когда им естественным образом передается управление. Секции отладки выполняются, как только в работающей программе происходит обращение к определенным идентификаторам, именам-файлов или именам-процедур. Секции отладки также не включаются при окончательной компиляции готовой программы.

Компиляцией строк и секций отладки управляет специальная фраза в параграфе SOURCE-COMPUTER (ИСХОДНАЯ-МАШИНА). Эта фраза добавляется в этот параграф следующим образом:

SOURCE-COMPUTER. имя-машины [WITH DEBUGGING MODE].

Когда задана фраза отладки WITH DEBUGGING MODE (В РЕЖИМЕ ОТЛАДКИ), компилируются все строки и секции отладки. Если фраза WITH DEBUGGING MODE отсутствует, все строки и секции отладки рассматриваются как строки комментариев. Строка отладки идентифицируется наличием буквы «D» в поле индикатора. Таким образом, в поле индикатора (обычно в 7-ой позиции) могут появиться четыре отметки: «-» для строки продолжения, «*» или «/» для строки комментария и «D» для строки отладки. Секции отладки идентифицируются тем, что они присутствуют в декларативной части раздела процедур и в начале каждой секции отладки помещается оператор USE FOR DEBUGGING (ИСПОЛЬЗОВАТЬ ДЛЯ ОТЛАДКИ).

Даже когда присутствует фраза WITH DEBUGGING MODE, задающая компиляцию строк и секций отладки, полученные в результате этой компиляции команды рабочей программы не выполняются до тех пор, пока не будет включен определенный реализацией переключатель отладки. Такое включение могло бы осуществляться

с помощью физического переключателя на пульте вычислительной машины или с помощью логического переключателя, активируемого считыванием специальной управляющей карты. В любом случае программисту предоставляется дополнительная возможность выбора, позволяющая либо использовать, либо игнорировать операторы отладки во время исполнения программы.

Если используются секции отладки, они должны записываться все вместе сразу же после заголовка DECLARATIVES. Все процедуры внутри этих секций могут обращаться только к другим процедурам секций отладки. Обращения к каким-либо операторам, расположенным вне секций отладки, не допускаются. Формальное определение секции отладки таково:

```

имя-секции SECTION.
      USE FOR DEBUGGING ON
      { ALL REFERENCES OF идентификатор }
      { имя-процедуры
        имя-файла
        ALL PROCEDURES } ...

```

Секций отладки может быть сколько угодно, и каждая из них должна иметь отдельный оператор USE FOR DEBUGGING. Любой отдельный идентификатор, имя-процедуры или имя-файла могут появляться только в одном операторе USE FOR DEBUGGING. Фраза ALL PROCEDURES (ВСЕХ ПРОЦЕДУРАХ) может быть задана только один раз. В этом случае ни одно отдельное имя-процедуры не может быть записано ни в одном операторе USE FOR DEBUGGING. Из секций отладки возможны обращения к специальному данному с именем DEBUG-ITEM (ДАННЫЕ-ОТЛАДКИ). Никакие внешние по отношению к секциям отладки операторы, включая операторы строк отладки, не могут обращаться к данному DEBUG-ITEM, а само это данное не должно описываться в разделе данных. Как и индексные данные, данное DEBUG-ITEM предусматривается самой системой, а не программой. Описание данного DEBUG-ITEM приведено ниже, при этом каждое элементарное имя-данного может использоваться отдельно, но только в секциях отладки:

01 DEBUG-ITEM.

05 DEBUG-LINE	PICTURE IS X(6).
05 FILLER	PICTURE IS X VALUE IS SPACE.
05 DEBUG-NAME	PICTURE IS X(30).
05 FILLER	PICTURE IS X VALUE IS SPACE.
05 DEBUG-SUB-1	PICTURE IS 9(4).
05 FILLER	PICTURE IS X VALUE IS SPACE.
05 DEBUG-SUB-2	PICTURE IS 9(4).

05 FILLER	PICTURE IS X VALUE IS SPACE.
05 DEBUG-SUB-3	PICTURE IS 9(4).
05 FILLER	PICTURE IS X VALUE IS SPACE.
05 DEBUG-CONTENTS	PICTURE IS X(60).

Значения этих данных устанавливаются автоматически при передаче управления секции отладки. Содержимое данного DEBUG-LINE (СТРОКА-ОТЛАДКИ) — это обычно номер, идентифицирующий конкретный оператор исходной программы и приведенный в распечатке программы, подготавливаемой компилятором. В данном DEBUG-NAME (ИМЯ-ОТЛАДКИ) содержится идентификатор, имя-файла или имя-процедуры, к которым произошло обращение, вызвавшее передачу управления секции отладки. При этом значения индексов или имена-индексов в данное DEBUG-NAME не включаются, они помещаются в данные DEBUG-SUB-1 (ИНДЕКС-ОТЛАДКИ-1), DEBUG-SUB-2 и DEBUG-SUB-3 в зависимости от их числа. Если индексы или имена-индексов отсутствуют, то эти данные заполняются пробелами. Последнее данное DEBUG-CONTENTS (ЗНАЧЕНИЕ-ОТЛАДКИ) содержит информацию о значениях, связанных с причиной передачи управления секции отладки.

Причина передачи управления	DEBUG-CONTENTS
Обращение к идентификатору	Содержимое идентификатора после выполнения обратившегося оператора
Обращение к имени-файла	Для оператора READ: полная только что считанная запись; для операторов OPEN и CLOSE: все пробелы
Обращение к имени-процедуры	Все пробелы, за исключением случая оператора ALTER (ИЗМЕНИТЬ), когда заносится имя измененной процедуры
Непосредственная передача управления процедуре с именем имя-процедуры	"FALL THROUGH" ("ПЕРЕХОД")

Автоматическая передача управления секции отладки и установка значений в данное DEBUG-ITEM происходят следующим образом:

1. Для идентификатора — непосредственно после любого выполняемого оператора, за исключением оператора WRITE, или непосредственно перед выполнением оператора WRITE.

2. Для имени-процедуры — непосредственно перед операторами GO TO, PERFORM или SORT или перед непосредственной передачей управления, или непосредственно после оператора ALTER.

3. Для имени-файла — после выполнения операторов OPEN или CLOSE и любого выполнения оператора READ, не приводящего к выполнению связанных с этим оператором повелительных операторов.

Предметный указатель

Адрес данных 64

Арифметические глаголы 103

— операции 144—181

Арифметический оператор ADD (СЛОЖИТЬ) 144, 147—149, 281—282

— — COMPUTE (ВЫЧИСЛИТЬ) 155—161

— — DIVIDE (РАЗДЕЛИТЬ) 144, 151—152

— — MULTIPLY (УМНОЖИТЬ) 144, 150—151

— — SUBTRACT (ОТНЯТЬ) 144, 149—150

Бит 79—81, 351

Блок-схемы составление 10—15

Буфер 244

Вызов программы 302—306

Выражения арифметические 155—159, 165

Выход из цикла 174

Вычислительной машины структура 63—66

Глоссарий 52—54, 99—100

Данное 9, 99

— групповое 86—89, 99

— индексное 329—331, 348

— нечисловое 84, 99

— числовое 85—86, 100

— элементарное 82—86, 100

Данные соответствующие 280

Данных обработки глаголы 102—103

— описание см. Описание данных

— описания различные 293—301

— совместное использование 306—309

— структуризация см. Структуризация данных

Декларативные секции 261—268, 439, 440, 474, 475

Доступ последовательный 247—248

— произвольный 392

- Заголовок 52
 - параграфа 52
 - раздела 52
- Задание по умолчанию 247
- Запись логическая 70—75, 99
- Запоминающие устройства 66—70, 391—392
- Запятая 29, 31, 229, 258, 322
- Зарезервированные слова 30—31
- Знак числа 99
- Значений представление 79—86
- Имен уточнение 277—280
- Имена мнемонические 228—232
 - одинаковые 280
- Имя катушки/пакета 108
 - машины 41—42
 - программы 29, 53
 - устройства 42, 53, 108
 - файла 42, 108—110
- Индексация 313, 316—320, 327—333
- Индексы 313, 316—321
- Исходная программа 9—62
- Кавычки 29, 31, 38
- Катушки/пакета имя 108
- Ключа относительного вычисление 408—419
- КОБОЛ, введение в программирование на языке 9—26
- Комментариев строки 36, 269—272
- Коммуникации средства 465—473
- Компилятор 9
- Литер строка 100, 125, 127—128, 132, 133—135, 137—138
- Литера 27—31, 53
 - буквенная 29, 53
 - буквенно-цифровая 30, 53
 - КОБОЛа 27—31, 53
 - отношения 30
 - пунктуации 29
 - специальная 30, 53
 - цифровая 29, 53
- Литерал 31—32, 53, 93—94
 - нечисловой 31, 32
 - числовой 31, 32
- Машинны имя 41—42
- Межпрограммные связи 302—310
- Мета-язык 48

Номер уровня 99, 273

Оборудование 63—75

Обработка файлов 182—185

— — непоследовательная 391—441

— — последовательная 228—268

Ограничитель, 444, 447, 452, 453

Оператор 32—34, 53

— АССЕРТ (ПРИНЯТЬ) 230—231

— ADD (СЛОЖИТЬ) 144, 147—149, 281—282

— ALTER (ИЗМЕНИТЬ) 476, 477

— CALL (ВЫЗВАТЬ) 302, 304

— CANCEL (ОСВОБОДИТЬ) 306

— CLOSE (ЗАКРЫТЬ) 110, 257, 258—259, 434, 476, 477

— COMPUTE (ВЫЧИСЛИТЬ) 155—161

— DELETE (УДАЛИТЬ) 406, 407, 424, 425

— DISPLAY (ВЫДАТЬ) 49, 184—185, 228, 229, 348—352

— DIVIDE (РАЗДЕЛИТЬ) 144, 151—152

— EXIT (ВЫЙТИ) 377, 440

— GO TO (ПЕРЕЙТИ К) 162, 336, 345, 367—370, 436, 477

— IF (ЕСЛИ) 162—163, 204, 331—332, 355—357, 361, 372

— INSPECT (ПРОСМОТРЕТЬ) 443, 445, 456—462

— MERGE (СЛИТЬ) 438—441

— MOVE (ПОМЕСТИТЬ) 113—117, 280—283, 297—298, 348, 353, 438, 443, 446

— MULTIPLY (УМНОЖИТЬ) 144, 150—151

— NEXT SENTENCE (СЛЕДУЮЩЕЕ ПРЕДЛОЖЕНИЕ) 163, 336

— OPEN (ОТКРЫТЬ) 105—111, 257—259, 394, 434—435, 476—477

— PERFORM (ВЫПОЛНИТЬ) 328, 330, 362, 436, 465

— — полный 379—390

— — простой 371—379

— READ (ЧИТАТЬ) 111—113, 144, 184, 236, 240, 244, 255—256, 264—265, 395, 396, 405, 406, 407, 425, 476

— RECEIVE (ПОЛУЧИТЬ) 467, 470

— RELEASE (ПЕРЕДАТЬ) 434, 437, 438

— RETURN (ВЕРНУТЬ) 434, 438, 440

— REWRITE (ОБНОВИТЬ) 395, 396, 406, 407, 425

— SEARCH (ИСКАТЬ) 328, 329, 330, 334—340, 361

— SEARCH ALL (ИСКАТЬ ОСОБО) 342—346

— SEND (ПОСЛАТЬ) 467, 470

— SET (УСТАНОВИТЬ) 328—332, 383

— SORT (СОТИРОВАТЬ) 432, 433, 434—437, 477

— START (ПОДВЕСТИ) 425

— STOP (ОСТАНОВИТЬ) 161—162, 304, 306, 344, 371

— STRING (СОБРАТЬ) 442—448

— SUBTRACT (ОТНЯТЬ) 144, 149—150

— UNSTRING (РАЗОБРАТЬ) 443, 445, 449—456

- Оператор USE (ИСПОЛЬЗОВАТЬ) 50, 263—265, 403, 425, 434, 474
— WRITE (ПИСАТЬ) 112—113, 144, 184, 232—235, 258, 264—265, 395, 396, 403, 407, 425
Операционная система 20—21
Операция логическая AND (И) 164, 165, 169—170, 362—366
— — OR (ИЛИ) 164, 165, 169—170, 362—366
— — NOT (НЕ) 164, 165—170, 362—366
Описание данных 63—100
— языка формальное 48—52
Отладка 20, 473—477
Отметки 35—39
Очереди сообщений 466
- Параграф 33, 44, 101—102, 184, 462
— DATE-COMPILED (ДАТА-ТРАНСЛЯЦИИ) 269—271
— DATE-WRITTEN (ДАТА-НАПИСАНИЯ) 269—271
— EXIT PROGRAM (ВЫЙТИ ИЗ ПРОГРАММЫ) 304, 377
— FILE-CONTROL (УПРАВЛЕНИЕ-ФАЙЛАМИ) 42—43, 54, 182, 183, 184, 249, 432, 433, 439
— I-O-CONTROL (УПРАВЛЕНИЕ-ВВОДОМ-ВЫВОДОМ) 250—255
— OBJECT-COMPUTER (РАБОЧАЯ-МАШИНА) 41—42, 43, 182, 183, 463
— PROGRAM-ID (ПРОГРАММА) 40—41, 43, 182, 183, 269, 270
— SECURITY (ПОЛНОМОЧИЯ) 269—270
— SOURCE-COMPUTER (ИСХОДНАЯ-МАШИНА) 41, 42, 43, 182, 183, 474
— SPECIAL-NAMES (СПЕЦИАЛЬНЫЕ-ИМЕНА) 229—232
Параграфы заголовков 52
Передача информации 101—143
Перемещение групповое 123—124
— элементарное 114, 124—125
Перемещения допустимые 137—139
Подчиненные процедуры 373
Поиск в таблице непоследовательный 342—347
— — — последовательный 334—342
— по образцу 212—219
Поля 35—39
Последовательная обработка файлов 228—268
Последовательность выполнения 103, 161—163
Предложение 33, 54, 101—102, 184
Присоединить 100
Пробел 29, 42, 322
Пробелами дополнение 124
Программы примеры 45—47, 118—122, 182—185, 185—227, 283—293
Программа главная 302—306
— исходная 20, 53
— на языке КОБОЛ 15—19
— подчиненная 302—306

Программа рабочая 20, 54

Программы вызов 302—306

— имя 29, 53

Процедуры повторяющиеся 172—175

Раздел КОБОЛа 33, 40—45, 54

— DATA DIVISION (РАЗДЕЛ ДАННЫХ) 18, 40, 43—44, 46, 71, 91—93, 112, 183, 184, 208, 229, 231, 234, 432, 439

— ENVIRONMENT DIVISION (РАЗДЕЛ ОБОРУДОВАНИЯ) 19, 20, 40, 41—43, 76, 182, 183, 208, 229—230, 234, 249

— IDENTIFICATION DIVISION (РАЗДЕЛ ИДЕНТИФИКАЦИИ) 40—41, 182, 269—272

— PROCEDURE DIVISION (РАЗДЕЛ ПРОЦЕДУР) 18—19, 40, 44—45, 101—104, 183, 184—185, 194, 204, 214, 234, 241, 261—262, 432, 464, 474

Раздела заголовков 52

Реализации имя 229

Редактирование вставкой 133—136

— — плавающей 135—136

— — фиксированной 133—135

— подавлением нулей 136—137

— с помощью перемещения 122—131

Редактирования категории 124—131

— категория буквенная 127—128

— — буквенно-цифровая 128—129

— — — редактируемая 129—130

— — числовая нецелая 125—127

— — — редактируемая 131—132

— — — целая 125—127

Рекурсивное определение 362

Связи межпрограммные см. Межпрограммные связи

Сегментация 462—465

Секции декларативные см. Декларативные секции

— заголовков 52

Секция 33, 462—464

— CONFIGURATION SECTION (СЕКЦИЯ КОНФИГУРАЦИИ) 182, 183

— FILE SECTION (СЕКЦИЯ ФАЙЛОВ) 18, 19, 43—44, 52—53, 91—92, 183—184

— INPUT-OUTPUT SECTION (СЕКЦИЯ ВВОДА-ВЫВОДА) 182, 183, 249, 252

— LINKAGE SECTION (СЕКЦИЯ СВЯЗИ) 308

— WORKING-STORAGE SECTION (СЕКЦИЯ РАБОЧЕЙ-ПАМЯТИ) 44, 91, 93, 182, 184, 238, 250, 355

Система управления сообщениями 467

Скобки квадратные 49, 50

— круглые 29, 31, 157—158, 159, 362

— фигурные 49, 50

Сортировка 429—433

Специальные возможности языка 442—477

Средства коммуникации 465—473

— отладки 473—477

Стандартные константы 93—96, 100

Статья 33, 54

— описания записи 86, 87—88, 91—94, 112, 272—274

— SELECT (ВЫБИРАЯ) 42—43, 76, 108, 182, 183, 245, 249, 403, 405, 422, 432, 439

Строки 35—39

Структуризация данных 269—310

Таблица 311—347

Таблицы таблиц 321—327

Терминал 466

Точка завершающая 29, 31, 41, 102, 112

Трансляции процесс 19—25

Устройства имя 42, 53, 108

— массовой памяти 391—398

— памяти с произвольным доступом 391—392

Умолчание 247

Управляющие карты 21—22

Условие знака 168, 366

— класса 169, 362, 366

— отношения 165—168, 363—364, 366

Условия составные 361—367

— VALUE 354—361

Файл 12, 70—75, 100

— входной (INPUT) 108, 110, 111

— выходной (OUTPUT) 110, 111

Файла имя 42, 108—110

Файлов обработка см. Обработка файлов

Файлы с организацией индексной 420—429

— — —относительной 399—407

Формат языка 27—39

Фраза 32—35, 54

— ACCESS MODE IS RANDOM (ДОСТУП ПРОИЗВОЛЬНЫЙ) 402, 405

— ACCESS MODE IS SEQUENTIAL (ДОСТУП ПОСЛЕДОВАТЕЛЬНЫЙ)
247—248

— ADVANCING (ДО/ПОСЛЕ ПРОДВИЖЕНИЯ) 233—235

— AFTER (ЗАТЕМ) 382, 383

— ALL PROCEDURES (ВСЕХ ПРОЦЕДУРАХ) 475

— ALTERNATE KEY (ДОПОЛНИТЕЛЬНЫЙ КЛЮЧ ЗАПИСИ) 423

— ASSIGN TO (НАЗНАЧИТЬ) 182

— AT END (В КОНЦЕ) 111, 197, 346

— BEFORE/AFTER (ДО/ПОСЛЕ) 458, 459—460

- Фраза BLANK WHEN ZERO (ПРОБЕЛ КОГДА НУЛЬ) 275
- BLOCK (В БЛОКЕ) 243—245
 - CORRESPONDING (СООТВЕТСТВУЮЩИЙ) 280—283, 299, 319—320, 437
 - COUNT (СЧЕТ) 450, 451, 453
 - DATA RECORDS (ЗАПИСИ ДАННЫХ) 239, 240
 - DELIMITED BY (ОГРАНИЧИВАЯСЬ) 446, 451, 452
 - DELIMITER IN (ОГРАНИЧИТЕЛЬ В) 451, 452
 - DEPENDING (В ЗАВИСИМОСТИ ОТ) 345, 367—370, 372
 - DUPLICATES (С ДУБЛИРОВАНИЕМ) 423
 - ELSE (ИНАЧЕ) 204
 - ERROR (ПРИ ПЕРЕПОЛНЕНИИ) 372
 - FILE STATUS IS (СОСТОЯНИЕ ФАЙЛА) 248—249, 250
 - FOR (С УДАЛЕНИЕМ) 259
 - FROM (ОТ) 113, 230, 382, 387, 395, 437
 - GIVING (ПОЛУЧАЯ) 145—146, 151, 184, 282, 436, 439, 440
 - INDEXED BY (ИНДЕКСИРУЕТСЯ) 335, 337
 - INTO (В) 255—256, 438
 - INVALID KEY (ПРИ ОШИБКЕ КЛЮЧА) 403, 405, 424
 - IS NOT LESS THAN (НЕ МЕНЬШЕ ЧЕМ) 425
 - JUSTIFIED RIGHT (СДВИНУТО ВПРАВО) 274
 - LABEL RECORD IS STANDARD/OMITTED (МЕТКИ СТАНДАРТНЫ/ОПУЩЕНЫ) 76, 110, 184, 236—237
 - MULTIPLE FILE TAPE (НА ОДНОЙ КАТУШКЕ) 251—252
 - NO DATA (НЕТ ДАННЫХ) 470
 - OCCURS (ПОВТОРЯЕТСЯ) 313, 314, 316, 323, 324, 328, 332, 335, 337, 338, 343, 430
 - ON SIZE ERROR (ПРИ ПЕРЕПОЛНЕНИИ) 145, 146—147, 152, 155
 - ORGANIZATION IS INDEXED (ОРГАНИЗАЦИЯ ИНДЕКСНАЯ) 420—429
 - ORGANIZATION IS RELATIVE (ОРГАНИЗАЦИЯ ОТНОСИТЕЛЬНАЯ) 399—407
 - ORGANISATION IS SEQUENTIAL (ОРГАНИЗАЦИЯ ПОСЛЕДОВАТЕЛЬНАЯ) 247—248
 - OUTPUT PROCEDURE (ПРОЦЕДУРА ВЫВОДА) 439, 440, 441
 - OVERFLOW (ПРИ ПЕРЕПОЛНЕНИИ) 447, 451
 - PICTURE (ШАБЛОН) 83—86, 91, 93, 114—117, 123—139, 144, 184, 194, 281, 315, 353, 345, 357, 446, 460
 - POINTER (УКАЗАТЕЛЬ) 445, 446, 450—451
 - RECORD CONTAINS (В ЗАПИСИ) 241
 - RECORD KEY (КЛЮЧ ЗАПИСИ) 423
 - REDEFINES (ПЕРЕОПРЕДЕЛЯЕТ) 296—300, 314, 320, 322, 323, 353, 387, 443
 - REEL (КАТУШКА) 258—259
 - REMAINDER (ОСТАТОК) 152
 - RENAMES (ПЕРЕИМЕНОВЫВАЕТ) 293—296, 322, 324, 387
 - REPLACING (ЗАМЕНЯЯ) 459—460, 461
 - RESERVE (РЕЗЕРВИРОВАТЬ) 246—247

Фраза **ROUNDED** (ОКРУГЛЯЯ) 145, 152, 155

— **SAME... AREA** (ОБЩАЯ ОБЛАСТЬ) 250—251, 358—359

— **SEPARATE CHARACTER** (ОТДЕЛЬНАЯ ЛИТЕРА) 276

— **SIGN** (ЗНАК) 274, 275—276

— **SYNCHRONIZED** (ВЫДЕЛЕНО) 352—353

— **TALLYING** (СЧИТАЯ) 450, 453, 461

— **UNIT** (ТОМ) 258—259

— **USAGE IS** (ДЛЯ) 81, 329, 348—351, 353, 442, 450, 457

— **USING** (ИСПОЛЬЗУЯ) 307, 436, 439

— **VALUE IS** (ЗНАЧЕНИЕ) 44, 93, 184, 238, 273, 299, 308, 314, 320, 359

— **VALUE OF** (ЗНАЧЕНИЕ) 77, 108, 236, 238—239, 354

— **VARYING** (МЕНЯЯ) 336—337, 382, 383

— **WHEN** (КОГДА) 344

— **WITH DEBUGGING MODE** (В РЕЖИМЕ ОТЛАДКИ) 474

Циклов организация 172—175

Языка формат см. **Формат языка**

Оглавление

Предисловие редактора перевода	5
Предисловие	7
Глава 1. Исходная программа	9
1.1. Введение в программирование на языке КОБОЛ	9
1.2. Языковой формат	27
1.3. Четыре раздела КОБОЛа	40
1.4. Пример программы	45
1.5. Формальное описание языка	48
1.6. Глоссарий	52
Глава 2. Описание данных	63
2.1. Оборудование	63
2.2. Требования к вычислительной машине	75
2.3. Представление значений	79
2.4. Групповые данные	86
2.5. Статьи-описания-записей	91
2.6. Примеры описаний данных	96
2.7. Глоссарий	99
Глава 3. Передача информации	100
3.1. Раздел процедур	101
3.2. Операторы OPEN-CLOSE	105
3.3. Операторы READ-WRITE	111
3.4. Операции перемещения	113
3.5. Примеры заданий на программирование и программы	118
3.6. Редактирование с помощью перемещения	122
3.7. Основной процесс редактирования: числовая редактируемая категория	131
Глава 4. Арифметические и логические операции	144
4.1. Операторы ADD и SUBTRACT	144
4.2. Операторы MULTIPLY и DIVIDE	150
4.3. Оператор COMPUTE	155
4.4. Управление последовательностью выполнения	161
4.5. Условия	163
4.6. Повторяющиеся процедуры	172
4.7. Примеры задач с решениями	176
Глава 5. Примеры программ и упражнения	182
5.1. Обработка файлов	182
5.2. Перепись файла карт в файл изменений	185
5.3. Обновление файла сотрудников	195

5.4. Обработка нарядов на работу	205
5.5. Поиск по образцу	212
5.6. Вычисления и выборочная обработка	219
Глава 6. Описание последовательного файла	228
6.1. Мнемонические имена	228
6.2. Описание файла	236
6.3. Размещение файла	245
6.4. Параграф управления вводом-выводом	250
6.5. Работа с файлом	255
6.6. Декларативные секции	261
Глава 7. Структуризация данных	269
7.1. Статьи комментариев	269
7.2. Дополнительные фразы описания данных	272
7.3. Уточнение имен	277
7.4. Вариант CORRESPONDING	280
7.5. Пример программы	283
7.6. Различные описания одного и того же данного	293
7.7. Межпрограммные связи	302
Глава 8. Таблицы	311
8.1. Общее представление о таблице	311
8.2. Использование индексов	316
8.3. Таблицы таблиц	321
8.4. Использование имен-индексов	327
8.5. Последовательный поиск в таблице	334
8.6. Непоследовательный поиск в таблице	342
Глава 9. Средства эффективного программирования на КОБОЛе	348
9.1. Фраза USAGE	348
9.2. Синхронизация	351
9.3. Условия VALUE	354
9.4. Составные условия	361
9.5. Модифицируемые передачи управления	367
9.6. Простой оператор PERFORM	371
9.7. Полный оператор PERFORM	379
Глава 10. Непоследовательная обработка файлов	391
10.1. Устройства массовой памяти	391
10.2. Файлы с относительной организацией	399
10.3. Вычисление относительного ключа	408
10.4. Файлы с индексной организацией	420
10.5. Сортировка	429
10.6. Операторы сортировки	434
10.7. Оператор MERGE	438
Глава 11. Специальные возможности языка	442
11.1. Оператор STRING	442
11.2. Оператор UNSTRING	449
11.3. Оператор INSPECT	455
11.4. Сегментация	462
11.5. Средства коммуникации	465
11.6. Средства отладки	473
Предметный указатель	478

УВАЖАЕМЫЙ ЧИТАТЕЛЬ!

Ваши замечания о содержании книги, ее оформлении, качестве перевода и другие просим присылать по адресу: 129820, Москва, И-110, ГСП, 1-й Рижский пер., д. 2, издательство «Мир».

Дж. Маджинис

ПРОГРАММИРОВАНИЕ НА СТАНДАРТНОМ КОВОЛЕ

Научный редактор Л. Н. Бабынина. Млад. научн. редактор А. Н. Сандерова. Художник Л. М. Муратова. Художественный редактор В. И. Шаповалов. Технический редактор Г. Б. Алюдина. Корректор Н. А. Гиря.

ИБ № 1233

Сдано в набор 10.10.78. Подписано к печати 18.01.79. Формат 60×90¹/₁₆. Бумага кн. журн. Гарнитура латинская. Печать высокая. Объем 15,25 бум. л. Усл. печ. л. 30,50. Уч.-изд. л. 29,20. Изд. № 1/9832. Тираж 39 000 экз. Зак. 3238. Цена 2 р. 30 к.

Издательство «Мир»
Москва, 1-й Рижский пер., 2.

Ордена Октябрьской Революции и ордена Трудового Красного Знамени
Первая Образцовая типография имени А. А. Жданова Союзполиграфпрома
при Государственном комитете СССР по делам издательства,
полиграфии и книжной торговли.
Москва, М-54, Валуевская, 28

жних
ктор

хурн.
изд.



ПРОГРАММЫ ПРОБЛЕМЫ НА СЛУЖБАХ ПОДРОБНО КОБОЛЕ